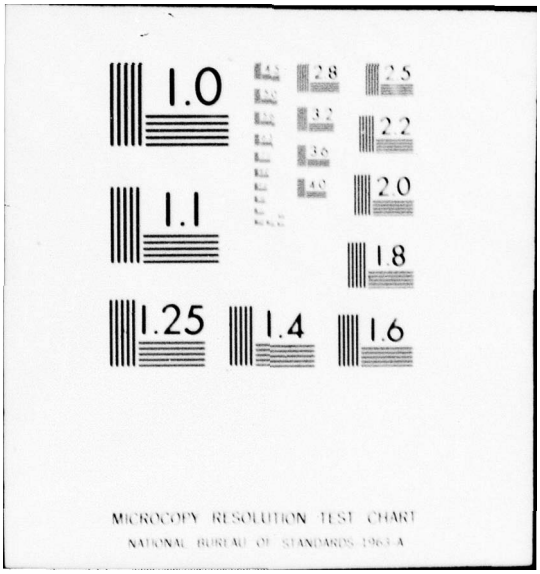
AD-A056 323 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 5/2
THE DESIGN AND IMPLEMENTATION OF INSTRUCTIONAL SOFTWARE INFORMA--ETC(U)
JUN 78 A YILDIRIM

UNCLASSIFIED

1 OF 2
AD
A056 323


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

AD A056323

LEVEL II

NAVAL POSTGRADUATE SCHOOL

Monterey, California

DDC
RECEIVED
JUL 18 1978
D

(9) Master's

THESIS,

(6) The Design and Implementation of Instructional Software Information System (ISIS).

by

(10) Ataman Yildirim

(11) June 1978

(12) 101 p.

THIS DOCUMENT IS BEST QUALITY PRACTICABLE.
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

Thesis Advisor: S. T. Holl

Approved for public release; distribution unlimited.

78 07 10 013
251 450

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Instructional Software Information System (ISIS)		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1978 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ataman YILDIRIM, LT, Turkish Navy		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE June 1978 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, CA 93940		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computer software
PDP-11 computer
INGRES data base application
QUEL query language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis describes the design, implementation and user interface for an Instructional Software Information System (ISIS).

The existing volume and increasing rate of growth of computer software production suggests the need for a catalogue procedure to help programmers find existing software to reduce redundant programming.

CONT'D

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

UNCLASSIFIED

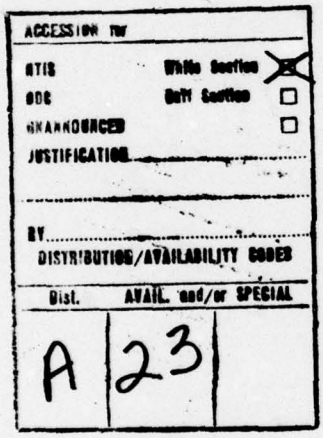
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

78 07 10 013

cont. → The purpose of the ISIS is to provide an online software catalogue which does not require either prior instruction or familarity with keyword lexicons.

Using ISIS, a user may record the characteristics of new software, and make searches for existing software by specifying its characteristics. Characteristics are specified by selection from a succession of menus.

ISIS is implemented on a PDP-11 computer operating under the UNIX operating system using the INGRES data base management system. It is written in the QUEL query language embedded in the programming language C.



Approved for public release; distribution unlimited

The Design and Implementation of Instructional Software Information System (ISIS)

by

Ataman YILDIRIM
Lieutenant, Turkish Navy
Turkish Naval War Academy, 1969

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE, 1978

Author : A. Yildirim

Approved by: D. Holl
Thesis Advisor

R. Ferrari
Second Reader

Chairman, Department of Computer Science

A. Schrady
Dean of Information and Policy Sciences

ABSTRACT

This thesis describes the design, implementation and user interface for an Instructional Software Information System (ISIS). The existing volume and increasing rate of growth of computer software production suggests the need for a catalogue procedure to help programmers find existing software to reduce redundant programming.

The purpose of the ISIS is to provide an online software catalogue which does not require either prior instruction or familarity with keyword lexicons. Using ISIS, a user may record the characteristics of new software, and make searches for existing software by specifying its characteristics. Characteristics are specified by selection from a succession of menus. ISIS is implemented on a PDP-11 computer operating under the UNIX operating system using the INGRES data base management system. It is written in the QUEL query language embedded in the programming language C.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to all those who provided background material and shared their experience in this thesis, and in particular to Dr. C. E. Irvine for her help with the C language and the UNIX operating system. I am deeply grateful to my thesis advisor LCDR Stephen HULL for his guidance, timely advice and help throughout this work. Finally, a very special thanks to my wife, Ayten, for her patience and understanding.

TABLE OF CONTENTS

I. INTRODUCTION.....7

A. INTRODUCTION TO THE PROBLEM.....7

B. POSSIBLE SOLUTIONS.....8

II. APPROACH TAKEN.....9

A. BASIC INITIAL SPECIFICATIONS.....9

B. DATA STRUCTURE.....10

1. Tree Type Structure.....10

2. Table Type Structure.....11

3. Hybrid Structure.....12

C. LANGUAGE SELECTION.....13

D. CLASSIFICATION OF SOFTWARE.....14

III. DESIGN FEATURES.....16

IV. CODING AND TESTING PHASES.....19

V. CONCLUSION.....21

A. SYSTEM CAPABILITIES.....21

B. PROBLEMS WITH THE SYSTEM.....22

C. POSSIBLE OTHER USAGE AREAS.....23

APPENDIX A : USER'S REFERENCE MANUAL.....24

COMPUTER PROGRAMS.....59

LIST OF REFERENCES.....100

INITIAL DISTRIBUTION LIST.....101

I. INTRODUCTION

A. INTRODUCTION TO THE PROBLEM

Today in the computer world there is an enormous amount of software, written on many different subjects. Sometimes it is feasible to adopt a program on hand to a given problem; sometimes it is not. Even when new software must be created, the job can often be made easier if advantage is taken of existing software tools or of example programs performing analagous functions.

The existing volume and increasing growth rate of computer software demands effective, easy to use catalogue procedures. When a programmer is tasked to write a new program, he should be able to begin by verifying that software performing the indicated tasks does not already exist.

Many computer firms supply on line or off line catalogued subroutine libraries. But these library catalogues are not general; they are catalogues on specific subjects. A general all-encompassing catalogue is desired, and it should be one which can be conveniently extended to include new software written or acquired locally.

C. POSSIBLE SOLUTIONS

Solutions to the problem can take two forms; an offline catalogue, or an online catalogue. After preparation, an offline catalogue is hard to update. An online catalogue is easy to use and maintain, but terminal access is required. In an educational environment this approach is especially appealing; it encourages the students to use software tools and gives them the ready access to a broad range of software which they would otherwise have only after gaining considerable programming experience. Learning to take advantage of software tools is most important for computer science students, especially during the later quarters of the curriculum.

II. APPROACH TAKEN

A. BASIC INITIAL SPECIFICATIONS

After the decisions to use an online catalogue was made, the following questions needed to be addressed:

a. What functions must the system perform ?

b. How should the system communicate with the user ?

The answers to these questions are really basic functional specifications:

The system has to search system resident listings according to user commands. The user has to be able to add new listings and also to be able to change his listings.

The system has to be very easy to use: the user should have no trouble even the first time he uses it. The approach taken to accomplish this was to make the system entirely self prompting. No command vocabulary is used; instead, the system asks questions and provides, with each question, a complete menu of responses. This philosophy is incompatable with the keyword approach which is a common feature of automatic cataloging systems.

5. DATA STRUCTURES

At this point the question of how to design the structure of the data representing the listings was addressed. The characteristics of importance are:

(1). Easy user understandibility, or else usability without understanding.

(2). The ability to quickly identify listings whose characteristics include those specified by the user.

(3). Reasonable storage density.

Listing lookup was expected to occur much more frequently than the creation of new listings or the modification of old ones; consequently the rapidity and complexity of listing structure updating was not considered to be an important criterian for data structure choice.

The following alternatives were considered:

1. Tree type structure

In a tree form software directory, the leaves would be individual listings, and the nodes would each be a common ancestor of all listings with a common characteristic.

One finds listings by starting at the root and making choices at each node, until reaching the listings desired.

Advantage:

The mechanism is easy to understand.

Disadvantages:

(1) User is forced to answer the questions in a particular order.

(2) Pointer implementation is troublesome.

2. Table type structure

One table with two dimensions is created; rows would correspond to program listings; columns to characteristics. When a listing has a given characteristic, a mark is placed in the appropriate cell in the table.

Advantages:

(1) Easy for user to comprehend.

(2) User can specify characteristics in any order.

Disadvantages:

(1) A large volume of storage is required.

(2) A variable dimension table is required because both listings and characteristics may be added as the database grows.

(3) Program purpose classification appears to be most conveniently organized as a tree.

(4) User needs to know the characteristic vocabulary in advance.

3. Hybrid structure

A compromise between the tree and tabular approaches was sought in the form of a hybrid structure. Here the user may picture a table of the form explained above, and he may think of the search process as one of choosing blocks of columns to omit and columns to save, successively refining the choices, until the listings are narrowed down to a small number satisfying the characteristics he needs. The selection of characteristic columns is organized in a tree, though the user need not be aware of this. His first choice - the first level of the tree - is this: from among all of the independent categories of software classifications he chooses those from among whose subdivisions he wishes to choose. For example a user whose goal is to find a sort for the IBM 360 will look over the first menu, containing "language", "hardware",... and will specify hardware for further clarification, and may ignore language if any language will do. The subdivisions of each of the chosen major subdivisions of each of the chosen categories are presented for review - then the minor subdivisions - and so forth, and the user may refine his choice along any direction as long as he wishes until further subdivisions are exhausted.

The physical form chosen for this structure was a tree where each node contains a table of names of its subordinate node tables and names of software listings which have the characteristics of that node but are too general to be listed among the node's subordinate nodes. (Notice that a given listing will typically be referenced in several different parts of the characteristics tree, most likely once under each major subcategory of the root.)

The relational database model offered an easy mechanism for maintenance of these node tables.

At this point it was decided to use the INGRES database management system available on the NPS computer laboratory PDP-11 computer under the UNIX operating system.

C. LANGUAGE SELECTION

The system programs are written in QUEL (the INGRES query language) embedded in the programming language C.

D. CLASSIFICATION OF SOFTWARE

A plausible, intuitive software classification was sought but no existing system seemed to be entirely suitable; moreover, they differed widely suggesting that classification goodness may be in some degree relative to the environment is which the catalogue is to employeed. So it was decided to implement a variable type of classification which each user agency can configure as it sees fit.

Here is an example classification:

1- Computer software tools

- Operating system
- Compiler
- Language
- Time sharing
- Microprogramming
- Debugging aids

2- Data manipulating tools:

- Text editing
- Sort
- Merge
- Database
- Test data generators
- Data structures like stack, queue etc.

3- Mathematics

Math. functions

Probability

Operations research

4- Business applications

Engineering

Financial

Inventory

III. DESIGN FEATURES

ISIS is implemented as an INGRES database. A tree structure is simulated using relations. A small demonstration database is shown in figure 1.

The root of the tree is the "maintable" relation which is loaded at system installation time and can not be changed by the user. Maintable tuples show the main branches of the tree: these are the independent categories of software characteristics, like language and hardware. Every node of the tree is a relation with two fields, rid(record ID) and crname (class record name). Rid is used as a key to all actions. If a rid is equal to zero, it means there is a relation with that name which further subdivides the current relation. If a rid is greater than zero, it means crname is the record name itself. One listing may (and usually will) be found recorded in more than one branch (relation). There is no information about the listings on the tree relations; only rids are shown. All other information about a specific listing can be found in the recordfile and whr (where) relations, which are in the example shown in figure 2.

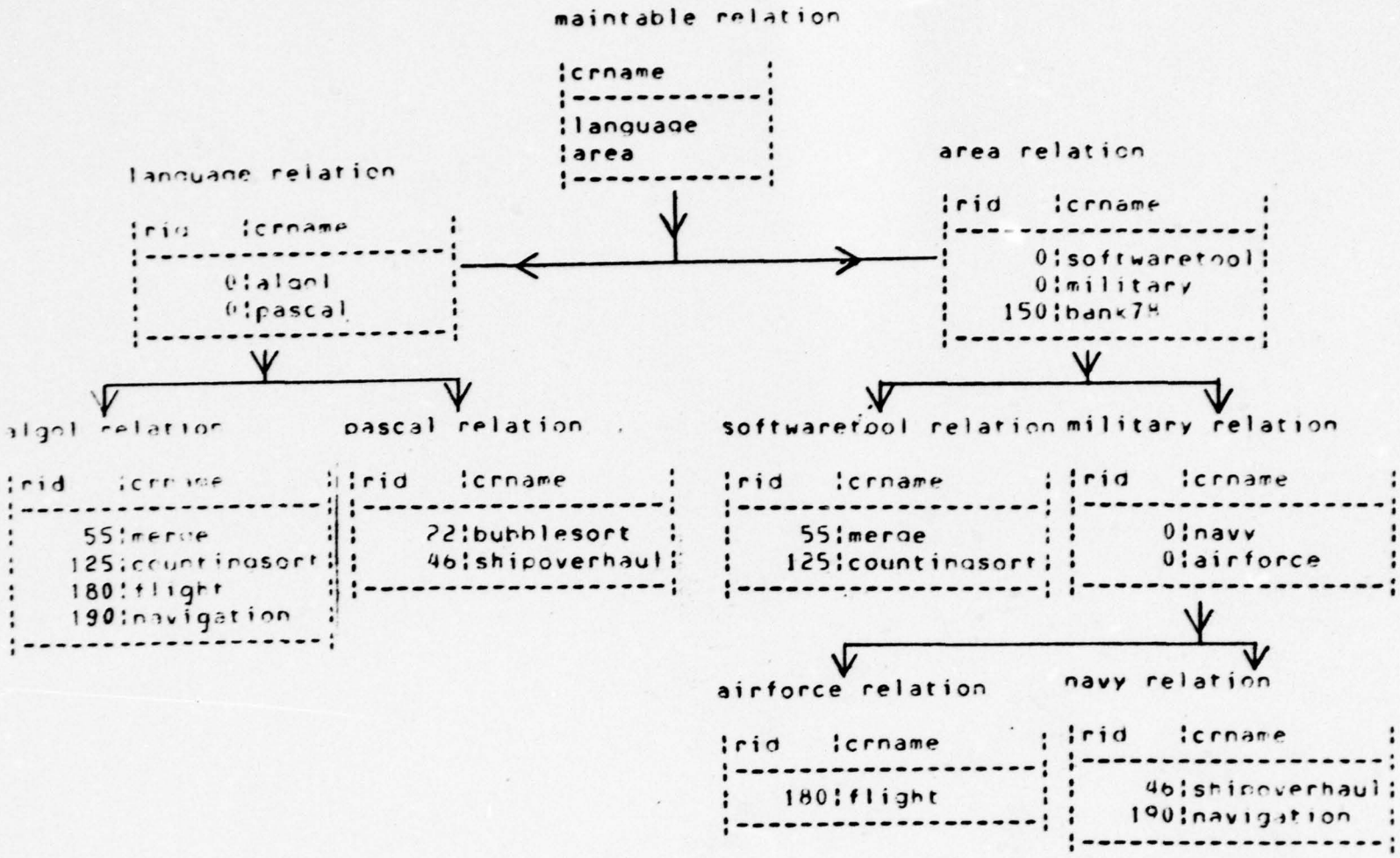


Figure 1. ISIS demonstration database "isis1".

recordfile relation

rid	rname	rdes	pswrd
22	bubblesort	sorts small files efficiently	a
46	shipoverhaul	pert analyses for shipyard	a
55	merge	merges 80 char long records according to key	a
125	countingsort	sorts 80 char long records according to key	a
150	bank78	checking account manipulation	a
180	flight	flight control processing	a
190	navigation	long range cruise	a

whr relation

rid	rpos
22	see /usr/James
46	see /usr/Robert
55	see Jack in room Sp-555
125	see prof. Smith
150	look at /usr/honey/tools
180	see Mike in room I-150
190	see Dennis in room I-260

Figure 2. Example of "recordfile" and "whr" relations.

IV. CODING AND TESTING PHASES

The ISIS system was designed in a top down manner. First the main frame of the system was designed, coded and tested. Then the primary function of the system, the search program, was designed, coded and tested. As the search program becomes large, some problems began to surface. The C compiler begin to print errors about undeclared variables which were in fact declared. At this point the output of EQUEL precompiler was checked and it was seen that similar statements in different parts of the source code were translated into C as different codes. When the source was divided into smaller files, the EQUEL precompiler seemed to work correctly.

After compilation was accomplished the testing began to take much more time than anticipated. The author was unprepared for the deliberateness with which INGRES functioned.

When run time error messages surfaced, it was difficult to find out what exactly was wrong because two compilers and the INGRES system were all involved.

EQUEL does not allow declaration of integer arrays. For this reason the integer array used as the input buffer was declared as a C variable and then, when required, each value was taken one by one from the C array into an EQUEL variable for use.

The search program is composed of approximately 1000 lines. It may cause interactive problems if run concurrently with add, delete and modify programs. The search routine is the key component of ISIS. It is possible to operate a reduced ISIS by running just the search function, maintaining the database (adds, modifications and deletions) by using the INGRES query language QUEL.

V. CONCLUSION

A. SYSTEM CAPABILITIES

The system has the following capabilities:

a. Search:

The system searches for resident listings according to user specified qualifications. The user can choose up to 25 qualifications for a given search.

b. Add:

Any user can add a listing; there is no need to get "add" permission from the ISIS manager. The principle restriction for addition is that the depth of the tree can be a maximum of around 20 arcs. When a listing is created, the listing "owner" assigns a one letter password.

c. Modify:

The owner of a listing can modify either the listing description and/or the "where found" information.

d. Delete:

The owner of a listing can delete his listing.

Currently the ISIS system can has a maximum of 32767 listings because listings are assigned 2 bytes unique ID numbers. By changing all "rid" attributes to i4 in relation declarations, the system could hold up to 2,174,483,647 listings.

Further information may be found in the user's manual.

B. PROBLEMS WITH THE SYSTEM

Only one user should use the search program or add, delete and modify program at a given time. If more than one user is using one of these, ISIS may not work correctly because some temporary ISIS information is stored in shared INGRES relations.

System response is very slow even when the listing database is miniscule. This is apparently a characteristic of the INGRES system.

It is impossible to print a hard copy of the system listings in catalogue form at present. Each branch point is embodied in a relation and all relations can be printed but it is hard to extract a catalogue form from them. By printing "recordfile" and "whr" relations all listings in the system can be seen as a list form.

It would be preferable to use a more efficient, faster database system to implement ISIS, or not to use a database system at all.

C. POSSIBLE OTHER USAGE AREAS

The ISIS system was designed with software cataloguing in mind; however the system could be used in many different applications, for example, the assignment of personnel to jobs. Personnel characteristics could be put into relations and when a person is needed for a specific job, he can be found through the search process.

APPENDIX A

USER'S REFERENCE MANUAL

The purposes of this manual are threefold:

1. To introduce the program to facilitate maintenance.

2. To provide information to the ISIS system manager to enable him to keep the system efficiently configured.

3. To provide the user with more detailed knowledge of the system and to list the error conditions.

Running the system is very easy and generally no previous experience is required. The first time user should not need to read this manual; he should simply log in as "isis" and then type "go".

TABLE OF CONTENTS

I. INTRODUCTION.....25

II. ISIS PROGRAMS.....26

A. GENERAL VIEW.....26

B. SEARCH PROGRAM.....28

1. General view.....28

2. Specifications of search program.....32

C. ADD-DELETE-MODIFY PROGRAM.....36

1. General View.....36

2. Specifications of Add-Delete-Modifiy Prog..40

III. SYSTEM GENERATION.....42

A. DATABASE SETUP.....42

B. COMPILATION OF PROGRAMS.....45

C. SIMPLE TUTORIAL.....47

D. ERROR MESSAGES.....57

I. INTRODUCTION

Instructional Software Information System (ISIS) is an online catalogue especially designed for computer software.

Using ISIS a user may record the existence and characteristics of a new program, and may make searches for existing software by citing the characteristics desired. Characteristics are specified by selection from a succession of menus.

ISIS is implemented on a PDP-11 computer operating under the UNIX operating system using the INGRES data base management system. It is written in the QUEL query language embedded in the programming language C.

II. ISIS PROGRAMS

A. GENERAL VIEW

The system is composed of three programs:

1. Main Program (isismain.c)

This program is written in the language C. Upon login as "isis", main is executed automatically. Then, according to user instructions, control is transferred to one of the programs in figure 3. Isismain.c is very straightforward and needs no explanation.

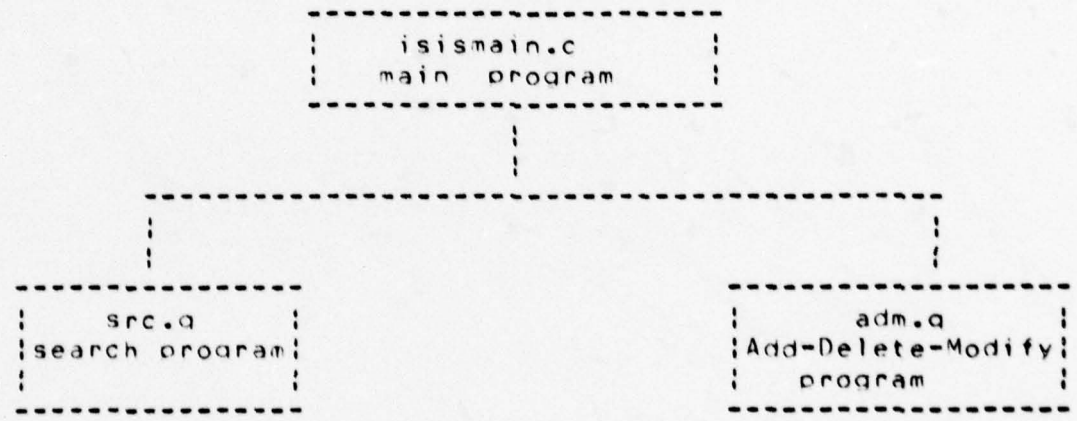


Figure 3. ISIS program heirarchy

2. Search Program (src.q)

This program is written in QUEL (the INGRES query language) and C. It does a search of ISIS software listings according to user qualifications and displays the results.

3. Add-Delete-Modify Program (adm.q)

This program is also written in QUEL and C. It allows a user to add new listings to the system. It also allows a listing originator to modify or delete an existing listing.

4. Data Structures

The principle ISIS data structures are described below:

a. The relation "recordfile" contains a master file of the listings, with each identified by name and unique ID number (RID).

b. Information about where to find the software corresponding to the listings is in the "whr" relation, identified by unique ID number.

c. The main software characteristic classes are the tuples of the "maintable" relation.

d. A variable set of "node" relations embodies the tree structure.

Each node relation contains:

1. Listing names and ID numbers.

and/or

2. The names of dependent node relations, identified by RID's of zero.

B. SEARCH PROGRAM

1. General View

The search program searches for listings of existing software meeting the characteristics specified by the user. The user specifies these by answering questions posed by ISIS.

System execution is slow even when the listing database is short; to prevent a long search over a short list, the user may be willing to scan a list of all of the records in the system. For this reason there is an option to display all records at the beginning.

A general flowchart of the search program is given in figure 4. After the user specifies some qualifications, the system replies how many listings meet the qualifications; the user may then either see the listings which meet his qualifications or he may change his qualifications.

After one search has been finished, the user may continue with other searches or he may exit from the system.

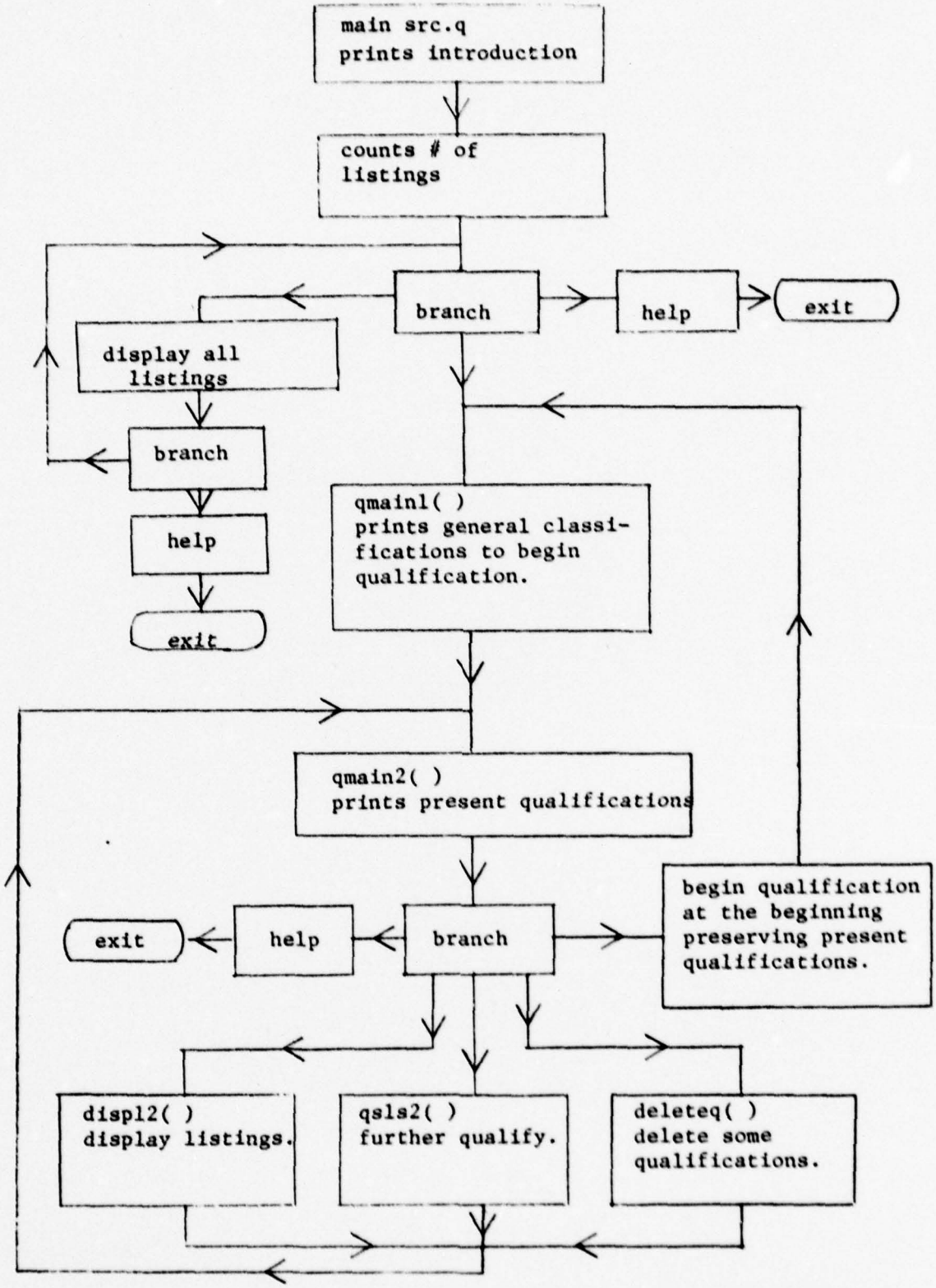


Figure 4. The general flowchart of search program src.q.

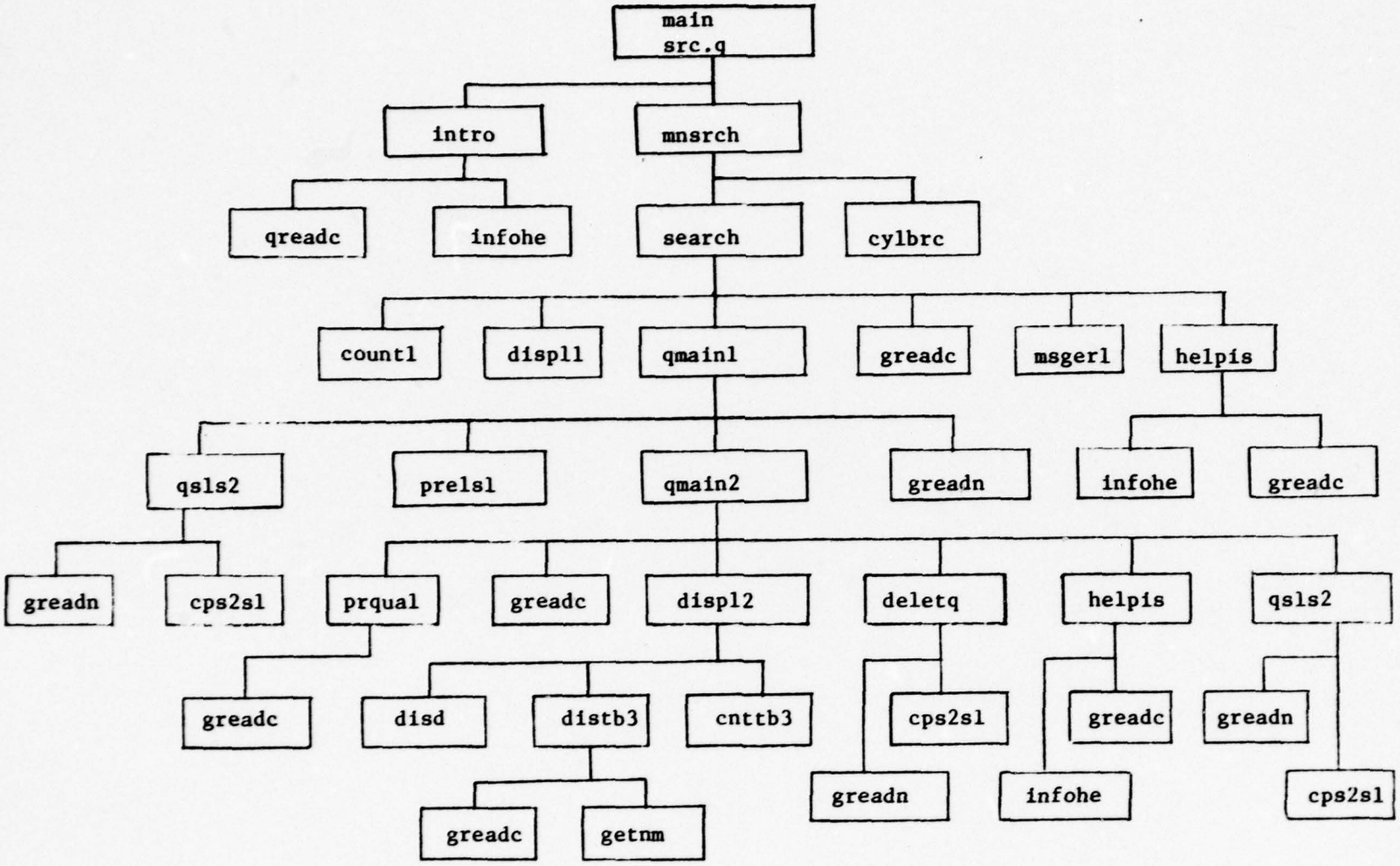


Figure 5. The functional diagram of the search program.

The functional diagram of the search program is shown in figure 5.

The source code for the search program is contained in three files.

a. File src.a

The code in this file performs the actual search; it is composed of the following functions;

main : calls introd and mnsrch functions.

intro : prints the introduction and invokes INGRES using the ISIS database.

mnsrc : calls the search routine, then exits ISIS.

search: monitors the search process by calling functions.

countl: counts number of listings in the system.

displl : displays all listings in the system.

mainl: monitors the qualification process.

prels1: puts relations onto stack 1.

qs1s2 : one by one gets a relation name from stack 1, qualifies it and puts results onto stack 2.

main2: monitors the rest of the qualification.

cps2s1: copies stack 2 to stack 1.

prqual: prints current qualifications.

deletq: deletes qualifications from stack 1.

displ2: displays listings according to qualifications.

b. File encsrc.a

This file is included in src.a at compile time.

The reason the search program is separated into two files, src.a and encsrc.a, is that the EQUEL precompiler does not seem to work correctly for long source files.

This file provides the following functions:

disd : displays D type qualifications.

cnttb3: counts table3 tuples.

disth3: displays listings in table3.

getnm : gets a listing number from terminal.

c. File com.a

This file is common to both the search program and the Add-Delete-Modify program. It provides the following functions:

cylbrc: evaluates the first usage of search program.

helpis: prints information helpful to a user who has "lost his way".

greadc: gets a character from terminal.

greadn: gets a maximum of ten integer numbers from terminal.

msgerl: prints an error message.

infohe: prints ISIS information of a general nature.

2. Specifications of the Search Program

A search is performed by specifying the characteristics of the sought for software. As the search continues, these characterictics are successively refined.

Search is done on a simulated tree structure. Every node of the tree is a relation. The root of the tree is the "maintable" relation. In order to begin the search process, a user must choose at least one branch.

The qualification process uses two qualification stacks, each 25 elements long, so the user can have a maximum of 25 qualifications at any one time. Stack 1 is used as the main stack; stack 2 is used as an auxilary stack when changing qualifications.

Each stack record has two fields:

a. qualification name

b. qualification type, which has value M , W or D. These codes are used to specify actions to be performed when the stack is processed.

M: this qualification shows miscellaneous tuples which are tuples with ID numbers greater than zero in the given relation.

W : Instruction to write all the tuples of a given relation which have an ID number greater than zero.

D : Instruction to display all tuples of this relation which are greater than zero; there exist here some tuples which have ID numbers equal to zero, meaning there are further subdivisions. So D, in effect, will cause branch records to be displayed also.

The display function (displ2) uses stack 1 records and according to the qualification type it employs different methods to find listings which meet all the specific qualifications.

To display M and N type qualifications, three table relations named table1, table2 and table3 are used. Table1 and table2 are used for listing ID's which meet one qualification. Note that the display process is an AND operation not an OR operation, so table3 identifies listings which meet all previously processed qualifications.

It is more difficult to display D type qualifications, which have branch relations, because it is also necessary to find out and evaluate all listings in the branches. For this reason, in addition to the three table relations mentioned above, two more relations named tabled1 and tabled2 are used. All listing rid's for a given qualification are saved in tabled1, and subbranch relation names are saved in tabled2. One tuple is taken from tabled2; if this tuple shows a branch, it is evaluated and the resulting listing ID's are put into tabled1; branches, if any, are put to tabled2 again, and the process continues until tabled2 is empty. At this time tabled1 has all listings which meet D type qualification.

Only one user at a time can employ this part of the program; otherwise these tables become confused. If this situation is suspected the user should check other terminals for a user using display; then wait and try again.

More than one user may be doing the qualification process at once; only display must be done serially.

C. ADD-DELETE-MODIFY PROGRAM

1. General View

This program is used to add new listings into the system and to delete or modify existing listings. A general flowchart of the program is given in figure 6 and the functional diagram in figure 7.

The source code of the program is composed of the following files:

a. File adm.c

This file performs add, delete, and modify actions and invokes functions in the other files.

File adm.c contains the following functions;

main : calls infohe and branch functions, and invokes the database.

branch: branches to add, delete or modify processes.

add : monitors add process.

getpar: gets parameters for a listing from terminal.

aprfwh: appends listings to recordfile and whr relations.

addrls: adds listings to relations by calling other functions.

putstr: puts relation name chosen by user on the stack.

pcurcl: prints current class and subcategories.

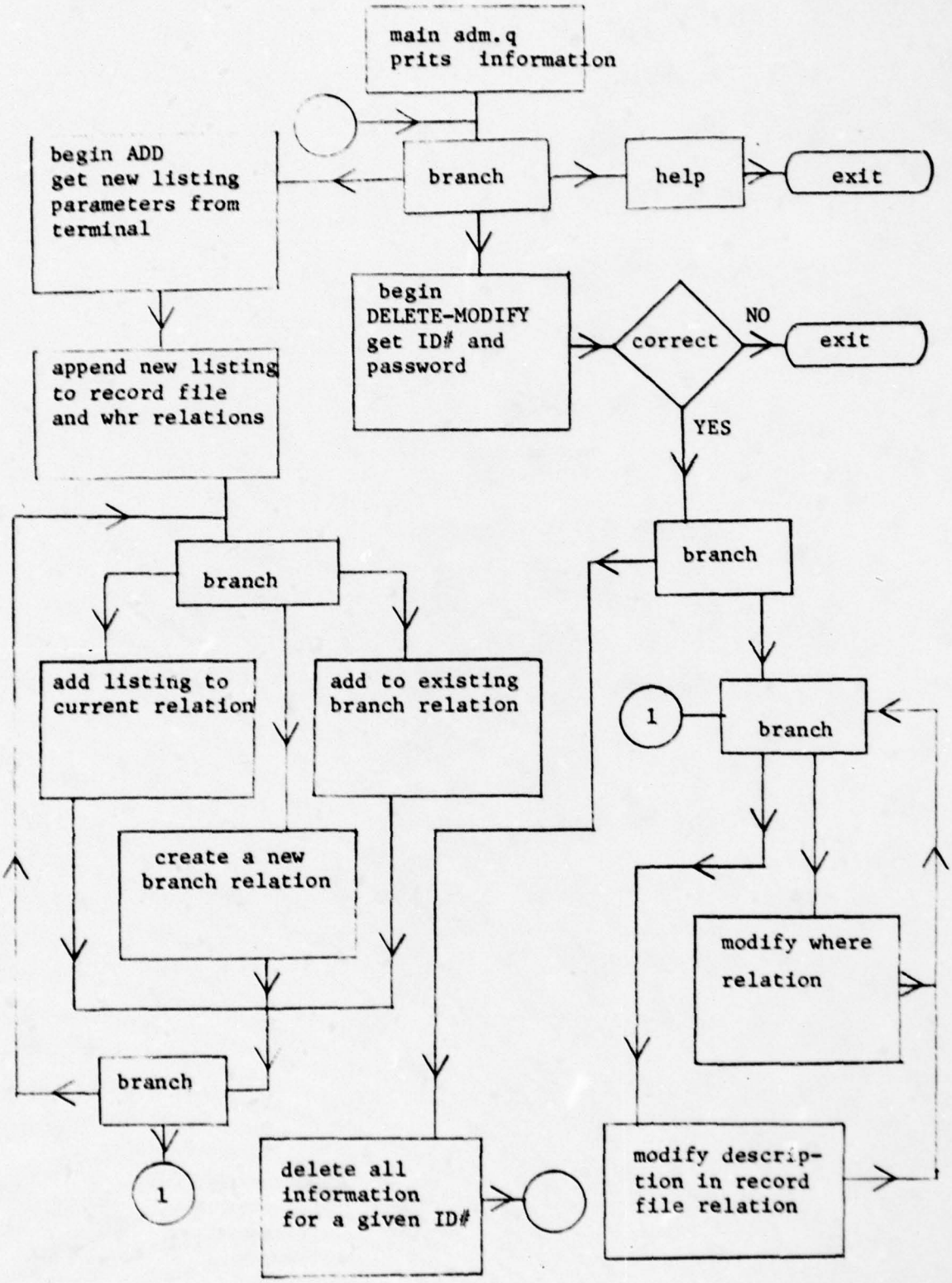


Figure 6. General flowchart of Add-Delete-Modify Program adm.q.

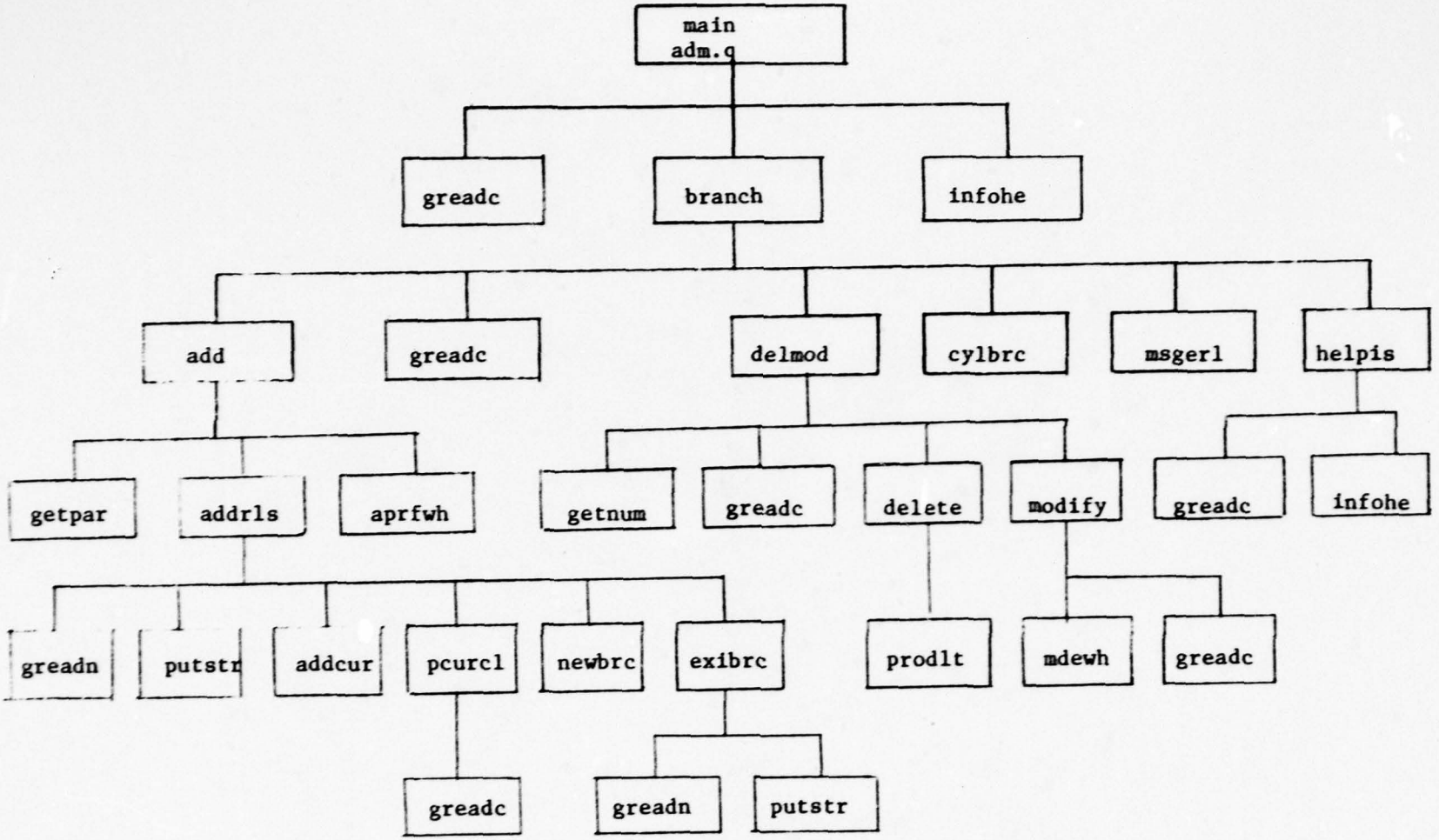


Figure 7. The functional diagram of Add-Delete-Modify program.

addcur: adds a listing to current relation.

newbrc: adds a listing to a newly created subcategory relation.

exibrc: adds a listing to one of the existing branches.

t. File encadm.q

This file is included in adm.q at compile time. These two files were split for the same reason that the search routine was divided; the EQUEL precompiler seemed unable to process both at once.

File encadm.q has the following functions;

delmod: gets parameters, and monitors delete and modify processes.

getnum: gets a listing ID number from the terminal.

delete: beginning part of the delete process.

prodlt: deletes a listing from a relation.

modify: beginning part of the modify process.

mdewh : modifies listing description and where information.

c. File com.q

This file contains functions common to the search and Add-Delete-Modify programs. For more information refer to the search program on page 32.

2. Specifications of Add-Delete-Modify program

The ADD process user can enter his listing into existing characteristic relations or he can create new branch relations and put the listing into them.

The ADD process can stack up to a maximum of 25 relation names. This does not mean the user is restricted to putting his listings into fewer than 26 relations, but rather that he can not have more that 25 further divisions identified and pending at any one time.

When a new relation is created as a new branch, the save command must be performed. This is done in file adm.q by the newbrc (new branch) function. The date supplied to the SAVE command must be advanced periodically by the ISIS manager; otherwise, when the date is reached, the relations may be inadvertently purged. It is probably best if the save date is identical to that used to save other relations.

DELETE and MODIFY processes at the beginning use the same delmod function to identify the owner of the listing which is to be deleted or modified. For this reason the user gives a listing ID number and the system checks to find out if there is any record with that number. Afterwards the password is checked; the password was set up during the ADD process when the listing was created; it is one character. For example it may be "." (fullstop) or any letter or digit. After the correct password is received, the delete or modify process can begin.

DELETE PROCESS:

This process deletes a listing from "recordfile", the master listing relation, and from "whr", the where found, relation, and begins to search the branch relations starting at the root. When the listing is found recorded in a relation, the subbranches of that relation are ignored.

When a listing is deleted from a relation, if there are no more listings in that relation, the relation is also destroyed. If the parent has no entries other than the destroyed relation then the parent relation is also destroyed; otherwise the child relation name is deleted from the parent.

The delete process employes the same stack used by the add process. The function dumwrt (dummy write), which prints the top 5 elements of the stack, was used in the testing phase of the program and was left in. It can be reactivated by removing the two slashes "//" (comment) previous to dumwrt(). By changing 5 to 25 in the "for" statement in dumwrt, all stack elements can be written during the delete and add processes.

MODIFY PROCESS:

It is only possible to modify listing description and where information. If the user needs to modify the tree entries for a listing he must first use DELETE to remove the listing entries and then use ADD to create revised ones.

III. SYSTEM GENERATION

This section explains how to install ISIS. ISIS requires the database management system INGRES which requires the UNIX operating system which runs on PDP/11 hardware.

A. DATABASE SETUP

The ACTUAL software database is named "isis0".

If a DEMONSTRATION database is desired, it should be named "isis1". A demonstration database is not necessary.

The database is created as follow.(The "%" sign is the UNIX prompt symbol)

First mount INGRES. At NPS this is done via a UNIX shell file, invoked as follows:

```
% /usr/ingres/sysmnt.sh
```

Now create database isis0:

```
%creatdb isis0
```

Invoke INGRES operating on the newly created database:

```
%ingres isis0
```

The INGRES prompt is "go *".

Now create the main listing and location relations and the "root" relations for the atribute tree:

```
go
create recordfile(rid=i2, rname=c12, rdes=c50, pswrd=c1)
create whr(rid = i2, rdes = c50)
create maintable(crname = c12)
```

Now "MAINTABLE", the root relation, must be loaded with the names of its direct descendents.

For example,

```
append to maintable(crname = "computer")
```

Now, "computer" is appended. Branch names have a maximum of 12 characters.

After filling the main table, create the corresponding node relations.

For example;

```
create computer(rid = i2, crname = c12)
```

Certain other relations which are used by ISIS as temporary storage during execution must also be created, as follows:

```
create  table1(rid = i2)
create  table2(rid = i2)
create  table3(rid = i2)
create tabled1(rid = i2)
create tabled2(crname = c12)
```

All the essential relations have now been created. DO NOT FORGET to "save" relations up to a reasonable date, for example 1 year ahead. For example:

save relationname until jan 30 1979

Repeat this command for each relation which has been created.

Now the database isis0 is ready. If it is desired to create the isis1 demonstration database, the procedure is repeated as above, except that the first command would be

%create isis1

8. COMPILATION OF PROGRAMS

To compile the main program in file isismain.c use the command

```
% cc isismain.c
```

Now rename the output "isis";

```
% mv a.out isis
```

To compile the search program:

First use the EQUEL precompiler for the files which are used by search.

```
% equel src.q
```

The output is in file src.c

```
% equel encsrc.q
```

The output is in file encsrc.c

```
% equel com.q
```

The output is in file com.c

These are all C files. Compile them using the EQUEL library:

```
% cc src.c -lq
```

The output is in file a.out; rename it "isissrc", as follows:

```
% mv a.out isissrc
```

"isissrc" is an executable file which, when invoked, performs an isis search.

To compile the Add-Delete-Modify program:

First, use the equel precompiler for the files which are used by this program.

```
% equel adm.q
```

The output is in file adm.c

```
% equel encadm.q
```

The output is in file encadm.c

The file com.c was already created during search compilation.

Now compile the program, using the EQUEL library.

```
% cc adm.c -lq
```

The output is in file a.out.

Rename it "isisadm", as follows:

```
% mv a.out isisadm
```

Now, "isisadm" is an executable file which, when invoked, performs isis add, modify, and delete operations.

Now remove unnecessary files:

```
% rm src.c encsrc.c com.c adm.c encadm.c
```

Isis is now ready to use.

C. SIMPLE TUTORIAL

This tutorial is prepared to give the user a general idea of how the system interacts with the user. This tutorial covers the search process only; it does not change the system resident data. It uses the ISIS demonstration database named "isis1" whose relations and linkage schema are shown in figure 1 on page 16.

```
-Type one of the following numbers in " "
  "1"  to SEARCH  ISIS  records
  "2"  to ADD new ISIS, DELETE or MODIFY existing ISIS
       records
  "3"  to QUIT ISIS

go >
1
     ISIS means Instructional Software Information System
          written at NPS
 *** when you need help, at some key point
     type "h" to get HELP ***
  good luck

  For more information type      "yes"
  Otherwise                      PUSH RETURN KEY go > yes
  This program is an online software dictionary
  There are two major programs:
  1. SEARCH program;
     User can find existing software by conversing with
the computer.
```

2. ADD, DELETE, MODIFY program:

User can advertise his own software by listing it with the system. Provisions for modifying and deleting a listing are included. Both programs are easy to use. They do not need previous experience. Just type answers after "go >" marks.

The programs are written in data sublanguage QUEL, supported by INGRES data base management system, and the programming language C.

```
 -Type one of the followings in " "
    "1"  to begin ACTUAL usage
    "2"  to begin DEMONSTRATION usage
go >
2 search in progress ...

There are 7 records about miscellaneous programs  in  the
system

  -Type one of the following numbers in " "
     "1"  to QUALIFY your need
     "2"  to DISPLAY all the records
go >
1

  -To QUALIFY your need
   Type one or more  of the following numbers in " "
        separating by commas
      You MUST type at least one
  "1"    language
```

```
   "2"    area
 go >
 1,2

   -To further QUALIFY   **  language    **
   Type one or more of the following numbers and/or letters
in " "
   "1" algol
   "2" pascal
   "n"  NOT to want qualify
 go >
 1

   -To further QUALIFY   **  area        **
   Type one or more of the following numbers and/or letters
in " "
   "1" softwaretool
   "2" military
   "m"  MISCELLENEOUS other than above
   "n"  NOT to want qualify
 go >
 1

Up to now, your qualifications are the following:
 1 -algol
 2 -softwaretool

   -Type one of the following numbers in " "
      "1"  to DISPLAY records which meet your
           qualifications
```

```
      "2"  to further QUALIFY
      "3"  to DELETE  some of qualifications
      "4"  to BEGIN QUALIFICATION at the beginning
         PRESERVING present qualifications
      "h"  HELP for other than above
go >
2

Up to now, your qualifications are the following:
 1 -algol        (has NO more qualification)
 2 -softwaretool(has NO more qualification)

   -Type one of the following numbers in " "
      "1"  to DISPLAY records which meet your
           qualifications
      "2"  to further QUALIFY
      "3"  to DELETE  some of qualifications
      "4"  to BEGIN QUALIFICATION at the beginning
         PRESERVING present qualifications
      "h"  HELP for other than above
go >
1
processing qualification: algol
processing qualification: softwaretool
At this point reduced to * 2 * records
There are * 2 * records which meet qualifications

   -Type one of the following numbers in " "
      "1"  to SEE records which meet qualifications
```

```
      "2"  to RETURN and CHANGE your qualifications
go >
1
ID #   record name          description
 55 ; merge         ; merges 80 char long records according
to key
 125 ; countingsort ; sorts 80 char long records  according
to key

  -Type one of the following numbers in " "
     "1"  to see WHERE information to find a specific
          record
     "2"  to RETURN
go >
1
  -Give only one of the ID numbers which have been printed
go >
125
ID #          where
 125          see prof.Smith

  -Type one of the following numbers in " "
     "1"  to see WHERE information to find a specific
          record
     "2"  to RETURN
go >
2

Up to now, your qualifications are the following:
```

```
1-algol        (has NO more qualification) 4 records
2-softwaretool(has NO more qualification) 2 records

  -Type one of the following numbers in " "
     "1"  to DISPLAY records which meet your
          qualifications
     "2"  to further QUALIFY
     "3"  to DELETE  some of qualifications
     "4"  to BEGIN QUALIFICATION at the beginning
        PRESERVING present qualifications
     "h"  HELP for other than above
ao >
3

  -Type
    one or more of the qualification numbers which have
been printed on the screen just before, separating by
commas
      OR
    "n"  NOT to want DELETE
ac >
2

Up to now, your qualifications are the following;
1-algol        (has NO more qualification) 4 records

  -Type one of the following numbers in " "
     "1"  to DISPLAY records which meet your
          qualifications
     "2"  to further QUALIFY
```

```
      "3"  to DELETE  some qualifications
      "4"  to BEGIN QUALIFICATION at the beginning
         PRESERVING present qualifications
      "n"  HELP for other than above
 go >
 4

   -To QUALIFY your need
    Type one or more  of the following numbers in " "
         separating by commas
       You MUST type at least one
   "1"    language
   "2"    area
 go >
 2

   -To further QUALIFY   **  area        **
   Type one or more of the following numbers and/or letters
in " "
   "1" softwaretool
   "2" military
   "m"  MISCELLENEOUS other than above
   "n"  NOT to want qualify
 go >
 2

Up to now, your qualifications are the following;
 1-algol       (has NO more qualification)
 2-military
```

```
   -Type one of the following numbers in " "
      "1"  to DISPLAY records which meet your
           qualifications
      "2"  to further QUALIFY
      "3"  to DELETE  some of qualifications
      "4"  to BEGIN QUALIFICATION at the beginning
         PRESERVING present qualifications
      "h"  HELP for other than above
 ao >
 2

   -To further QUALIFY   **  military    **
   Type one or more of the following numbers and/or letters
in " "
   "1" navy
   "2" airforce
   "n"  NOT to want qualify
 ao >
 1

Up to now, your qualifications are the following;
 1-algol        (has NO more qualification)
 2-navy

   -Type one of the following numbers in " "
      "1"  to DISPLAY records which meet your
           qualifications
      "2"  to further QUALIFY
      "3"  to DELETE  some of qualifications
```

```
      "4"  to BEGIN QUALIFICATION at the beginning
         PRESERVING present qualifications
      "h"  HELP for other than above
go >
1
processing qualification: algol
processing qualification: navy
At this point reduced to * 1 * records
There are * 1 * records which meet qualifications

  -Type one of the following numbers in " "
      "1"  to SEE records which meet qualifications
      "2"  to RETURN and CHANGE your qualifications
go >
1
 ID #   record name          description
 190 ! navigation   ! long range cruise

  -Type one of the following numbers in " "
      "1"  to see WHERE information to find a specific
           record
      "2"  to RETURN
go >
2

Up to now, your qualifications are the following;
 1-algol        (has NO more qualification) 4 records
 2-navy                                     2 records

  -Type one of the following numbers in " "
```

```
      "1"  to DISPLA  records which meet your
           qualifications
      "2"  to further QUALIFY
      "3"  to DELETE  some of qualifications
      "4"  to BEGIN QUALIFICATION at the beginning
         PRESERVING present qualifications
      "h"  HELP for other than above

go >
help

  Type one of the following numbers in " "
      "1"  to EXIT ISIS and return to UNIX
      "2"  to BEGIN ISIS at the beginning again
      "3"  to CONTINUE as you are
      "4"  to see more INFORMATION about ISIS

go >
1
ISIS has been  logged out
 goodbye, come again
```

D. ERROR MESSAGES

The INGRES system may print EQUEL error messages with error numbers and a little information during run time. Explanations of them can be found in reference 6. EQUEL errors arise when the system relations are not in proper form. For example in one relation there may be a tuple referencing a nonexistant dependent relation. This condition may occur if the computer goes down or if the user gives up in the middle of the add process. In order to fix the system, the relations must be checked by the manager using INGRES directly. (The UNIX command "RESTORE database name" should be issued whenever a crash occurs during an INGRES operation.)

There are other error messages which are printed by ISIS itself due to wrong arguments given by the user. These are all self explanatary, and give the user another chance to type a correct argument. In some places, if the user types the same wrong argument three times the system terminates.

```
main()  {  /* forks one of the two ISIS programs */

    int n;
    int i,j,k;

printf("-Type one of the following numbers in \" \"\n");
printf("      \"1\" to SEARCH ISIS listings\n\
\"2\"  to ADD new listings, DELETE or MODIFY existing\
ISIS listings\n      \"3\"  to QUIT ISIS\nqo >\n");
    n = getl();
    if(n == 1){
      if((i = fork()) == 0){
       execl("isissrc","isissrc",0);
       exit();
      }
      wait(&j);
    }
    if(n == 2){
        if((i = fork()) == 0){
            execl("isisadm","isisadm",0);
            exit();
        }
        wait(&j);
    }
    if(n == 3)
      exit();
}
```

```
// Istructional Software Information System (ISIS)
// INGRES database application
// This program does SEARCH portion of ISIS
// The other part of ISIS is in file adm.c which
// does ADD, MODIFY and DELETE functions
// This file has two includes:
//      1 .encsrc.q     2. com.q
// File com.c is common for both part of ISIS


// flag-erase: erase stack1 or not
   extern  char  ferase;
   extern  int   *erasep;


main()  {
    introd();  /* prints ISIS introduction */
    mnsrch();      /* main-search begins */
printf(" ISIS has been  logged out\n");
printf("  goodby, come again\n");
##   exit
}

introd() {  /* prints introduction */
##   char *is01, *intrq;
printf(" ISIS means Instructional Software Information\n\
          System, written at NPS \n\
 *** when you need help, at some key point type \"h\" to\
 get HELP ***\n  good luck\n");
  printf("\n    For more information type     \"yes\"\n\
    Otherwise                  PUSH RETURN KEY\ngo >\n");
    intrq = greadc();
    if(intrq == 'y')  infohe();
  again:
    printf("  -type one of the followings in \" \"\n\
    \"1\"  to begin ACTUAL usage\n\
    \"2\"  to begin DEMONSTRATION usage\ngo >\n");
    is01 = greadc();
    if(is01 == '1') {
##   ingres isis0  // invokes ingres database named ISIS0
    }
    if(is01 == '2') {
##   ingres isis1  // invokes demonstration database ISIS1
    }
    if(is01 != '1' && is01 != '2')
    { printf("\nWrong argument; try again\n"); goto  again;
}
 } /* end introd() */
```

```
mnsrch() { /* main-search; monitors search routine */
##    char *rt;
      rt = 'e';
      cycle:      /* is used to begin ISIS again */
            rt = search();   /* call main search routine */
      if (rt == 'f')  rt = cylbrc();   /*  in file com.a */
      if (rt == 'b' || rt == 'c')  goto cycle;
      return;
}           /* end mnsrch() */

search() {
##    char *cs, *rts;
##    int  trec, ns;
      ns = 0; printf("search in progress ...\n");
      trec = count1(trec);
      printf("\n There are %d records about miscellaneous\
      programs in the system\n", trec);
     again1:
 printf("-type one of the following numbers in \" \"\n");
      printf("       \"1\"  to QUALIFY your need\n\
       \"2\"  to DISPLAY all the records");
     if (rts == 'd') {
     printf(" again\n");
     printf("  \"3\"  to EXIT ISIS and return to UNIX\n\
       \"4\"  to BEGIN ISIS at the beginning again\n");
      }
       else  printf("  \n");
      rts = 'f';
     again:
      printf("go >\n");
      cs = greadc();
      switch(cs) {
      case '1':
            rts = gmain1();
            break;
      case '2':
            rts = disp11();
            break;
      case '3':
            rts = 'e';
            break;
      case '4':
            rts = 'b';
            break;
      case 'h':
            rts = helpis();
            break;
      default:
            if (ns < 3)  { msger1(); ns++; goto again; }
               else  break;
     }
      if (rts == 'c' || rts == 'd')   goto again1;

      return(rts);
```

```
}     /*  end of search routine */


count1() { // returns # of records in recordfile relation
##    int  otrec;
      otrec = 0;
##    range of r is recordfile
##    retrieve (otrec = count(r.rname) )
     return(otrec);
}


disp11() { // display-first: executed if display is
           // requested before any qualification
##    char rrdes[51], rrname[13], *rtd1;
##    int   rrid;
      rtd1 = 'd';
   printf("     All the records are the followings;\n");
   printf("ID #     record name       description\n");
##    range of r is recordfile
##    retrieve (rrid=r.rid,rrdes=r.rdes,rrname=r.rname) {
    printf("%d %s %s %s %s\n",rrid," |",rrname," |",rrdes);
##    }
     return(rtd1);
}



 ////////////////////////////////////////////////////////////////

 //  These two structures are used as stacks
 //  in the qualification process

     struct st1tag {
##             char  st1elm[13];
##                   char *st1m;
##                   int   st1c;
                    } st1[25];

     struct st2tag {
##                   char  st2elm[13];
##                   char  *st2m;
##                   int   st2c;
                    } st2[25];

 ////////////////////////////////////////////////////////////////
```

```
qmain1() { // qualification-main: first time
    //  qualification is done and search
    //  continues in this function by calling
    //  other functions.
    struct  st1tag  *pst1;
    extern  char ferase;
##  char mmname[13], *rtqua, *rtq, *rts12;
##  int  nmnum;
    int nbuf[12], *np,  i, i1 /* not preceeded with ## */
    np = &nbuf[0];
    rtqua = 'f';  // return function if finished correctly
    ferase = 'Y';

 //  to use stack1 again without exited program;
 //  initialize emty
    pst1 = &st1[0];
    for(i = 0; i < 25; i++) {
     for(i1 = 0; i1 < 13; i1++)   pst1->st1elm[i1] = ' ';
       pst1->st1m = ' ';
       pst1++;
    } /* end for 1 */

 qaqain: // comes if want to qualify without erase st1
  printf(" \n   -To QUALIFY your need \n    Type one\
  or more of the following numbers in \" \"\n\t\
  separated by commas\n You MUST type at least one\n");
  // print maintable relation to begin qualification
    nmnum = 0;
##   range of m is maintable
##   retrieve (mmname = m.crname) {
    nmnum++;
    printf("   \"%d\"    %s\n", nmnum, mmname);
##   }
    /* gets input from the terminal and puts in array */
    rtq = qreadn(&nbuf[0]);  /* function reads inputs */
    if (rtq == 'e') { rtqua = 'e'; goto endqua;}

   // put relation name on stack1
    prels1( &nbuf[0]);
    pst1 = &st1[0];
    rts12 = qs1s2(&nbuf[0]);
    if(rts12 == 'e') { rtqua = 'e'; goto endqua; }
    rtqua = qmain2(&nbuf[0]);
       if(rtqua == 'b' && ferase == 'N')  goto qaqain;
    endqua:
          ;
    return(rtqua);
}    /*  end qumain  */
```

```
prelsl(arrp)   // put-relation-stackl: puts relations
 // corresponding to array onto stackl from maintable
    int  *arrp;  {
    extern  char ferase;
    extern  int  *erasep;

##   int  relnum;
##   char  tname[13];
##   int ii;
##   char *flwr, *flrun;
    struct  stltag  *stlp;  stlp = &stl[0];
    flrun = '1';
    tname[12] = '\0';
    if(ferase == 'N')  {  // needs adjust stackl pointer
 while(stlp->stlm=='D'||stlp->stlm=='M'||stlp->stlm=='W')
    stlp++;
    erasep = stlp;   /* to use in qmainl */
    ferase = 'Y';
    }
    else
      erasep = 0;
    relnum = 0;
##   range of t is maintable
       while (*arrp != 0) {
##    retrieve (tname = t.crname) {
    relnum++;
    flwr = 'n';

    if(*arrp == relnum)  {
    for(ii=0;(stlp->stlelm[ii]=tname[ii]) != '\0';ii++);
       stlp->stlelm[ii] = '\0';
       stlp->stlm = 'D';
       stlp++; arrp++; flwr = 'y';}
##   }
    if(flwr == 'n')
     if(flrun == '1')  flrun = '2';
     else {
     if(*arrp == 98)
       printf("wrong argument \"m\" is discarded\n");
      else
       if(*arrp == 99)
        printf("wrong argument \"n\" is discarded\n");
       else
        ;
       arrp++; flrun = '1';
     } /* end else */
    relnum = 0;
   } /* end  while */
return;
}    /* end prelst() */
```

```
asls2(nbufb)   // qualification-from stack1-to stack2:
      //  does qualification by st1 and puts on st2
      //  and copies st2 to st1 at end
    int  *nbufb; {
    struct  st1tag  *pstal;
    struct  st2tag  *psta2;
    extern  int  *eraseo;

##   int  rnum, i, eric;
     int  *nbufbt, twr;
##   char tmprel[13], tcrname[13], *flst2;
##   char *flsy, *flsn, *rtal2, *rtar;

       /* initializations of variables */
    tcrname[13] = '\0';
    rtal2 = 'f'; /* if asls2() executed correctly */
    pstal = &st1[0];   psta2 = &st2[0];

 // if desired copy present stack1 elements to stack2
    if(eraseo != 0) {
      while(pstal < eraseo) {
        for(i = 0; i < 12; i++)
           psta2->st2elm[i] = pstal->st1elm[i];
        psta2->st2m = pstal->st1m;
        pstal++;   psta2++;
      } /* end while */
     eraseo = 0;
    } /* end if */


  while(pstal->st1elm[0] != ' ')  {
     flsy = 'n'; flsn = 'n';
  for(i= 0; i<12; i++) tmprel[i] = ' '; tmprel[12] = '\0';


    /* get a relation name from stack1 */
      if(pstal->st1m == 'D')
           { for(i = 0; i < 12; i++)
             tmprel[i] = pstal->st1elm[i];
           }
      else {
   /* if M or W mark, copy it from stack1 to stack2*/
           for(i = 0; i < 12; i++)
              psta2->st2elm[i] = pstal->st1elm[i];
           psta2->st2m = pstal->st1m;
           psta2++;
           goto endwhi;
           } /* end else */

##   range of e is tmprel
##   retrieve (eric = e.rid) {
     if(eric == 0)  flsy = 'y';
     else   flsn = 'y';
##   }
```

```
// if no more subdivisions previously marked as D

    if(flsn == 'y' && flsv == 'n')
    {
//  put relation on stack2 with M mark

    for(i = 0; i < 12; i++)
       psta2->st2elm[i] = psta1->st1elm[i];
    psta2->st2m = 'M';
    psta2++;
  // all records in relation marked 'M' will be displayed
  // in qmain2 in display

    goto endwhi;
    } /* end if */

//  there may or may not be some subdivisions

  else {
  printf("\n -To further QUALIFY ** %s** \n",tmprel);
  printf("  Type one or more of the following numbers");
  printf(" and\/or letters in \" \"\n");
        }

    rnum = 0;
##   retrieve (tcrname = e.crname)
##   where e.rid = 0  {
    rnum++;
    printf("   \"%d\" %s \n", rnum, tcrname);
## }

  if(flsn == 'y')
  printf("  \"m\" MISCELLANEOUS other than above\n");

    printf("   \"n\"  NOT want to qualify \n");

    //  adjust pointer to point to beginning
    nbufbt = nbufb;
    rtar = qreadn(nbufbt);
    if(rtar == 'e') { rtal2 = 'e'; goto endal2; }

   // according to array values, put corresponding
   // relations on stack2


    while(*nbufb != 0)  {  /* while # 1 */
     if(*nbufb == 99)  // means 'n'; not want to qualify
       goto endwhi;  // at the end qsls2 ()

//  if array value other than 'n' or 'm'
  while(*nbufb != 98 && *nbufb != 0) { /* while #2 */
//   put name from relation on stack2 with D mark
```

```
        rnum = 0;   fwr = 0;
         while(fwr < 2) {    /* while # 3 */
           fwr++;  rnum = 0;
##          retrieve (tcrname = e.crname)
##          where e.rid = 0 {
           rnum++;
           if(rnum == *nbufb) {
            for(i = 0; i < 13; i++)
                psta2->st2elm[i] = tcrname[i];
                psta2->st2m   =  'D';
            psta2++; fwr = 0; nbufb++;
           }
##          } /* end where */
         }  /* end while # 3 */
       if(*nbufb == 98) continue;
         else {
               if(*nbufb != 0)
       printf("warning-wrong argument %d or n is \
               discarded\n", *nbufb); nbufb++; }
     }  /* end while # 2 */


     if(*nbufb == 98) { /* means miscellaneous */
  // put relation name on stack2 with 'M' mark
       for(i = 0; i < 13; i++) {
          psta2->st2elm[i] = tmprel[i];
          }
          psta2->st2m = 'M';   psta2++;
     }  /* end if(*nbufb == 98) */

     if(*nbufb != 0)   nbufb++;


   }   /* end while # 1 */

   endwhi:
     psta1++;

 }  /* end while at the beginning */
cps2s1();   /* copies stack2 to stack1 */

   enda12:
         ;

}  /* end as1s2(rta12); */
```

```
cps2s1()  {  /* copy-stack2-to-stack1  */
 //  erases stack1, copies stack2 to stack1, erases stack2

    struct   st1tag   *po1;
    struct   st2tag   *po2;

##   int   j, j1;

    po1 = &st1[0];      po2 = &st2[0];

 //  erase stack1, to begin copy with a clean stack1
    for(j = 0; j < 25; j++) {
          for(j1 = 0; j1 < 12; j1++)
               po1->st1elm[j1] = ' ';
          po1->st1m = ' ';
          po1->st1c = 0;
          po1++;
       }
    po1 = &st1[0];


 //  copy stack2 to stack1, and erase stack2
    j = 0;
    while(j < 25)  {  /* while # 1 */
   if(po2->st2m=='D' || po2->st2m=='W' || po2->st2m=='M')

          /* copy part */
        {  for(j1 = 0; j1 < 12; j1++)
           po1->st1elm[j1] = po2->st2elm[j1];
           po1->st1m   = po2->st2m;
           po1->st1c = po2->st2c;

           /* erase stack2 */
           for(j1 = 0; j1 < 12; j1++)
              po2->st2elm[j1] = ' ';
           po2->st2m   = ' ';
           po2->st2c  = 0;
         }
         else   j = 25;
      po1++;  po2++;  j++;
    }  /* end while # 1 */


    return;
} /* end cps2s1 */
```

```
qmain2(nbufpo)   /* qualification-secondary   */
    int  *nbufpo;  {
    extern  char  ferase;
    struct   stltag  *plc;

##   char  *q2rth, *qm2rt, *q2c;
    int   *nbftmp, k3;
    ferase = 'Y';
    nbftmp = nbufpo;
    plc = &stl[0];
    qm2rt = 'c';
 //  erase stlc before beginning to print
 //  any qualifications
    for(k3 = 0; k3 < 25; k3++) {plc->stlc = 0; plc++;}
    while(qm2rt == 'c')  {
    prqual();
  printf("\n ");
  printf("-Type one of the following numbers in \" \"\n\
   \"1\" to DISPLAY records which meet your \
   qualifications\n");
   printf("      \"2\"  to further QUALIFY\n      \"3\"\
   to DELETE  some qualifications\n");
   printf("     \"4\"  to BEGIN QUALIFICATION at the\
     beginning\n PRESERVING present qualifications\n");
   printf("   \"h\" HELP for other than above\ngo >\n");
    q2c = greadc();
    switch (q2c) {
    case '1':
         qm2rt = disp12();
         break;
    case '2':
          qsls2(nbftmp);
         break;
    case '3':
         qm2rt = delete(nbftmp);
         break;
    case '4':
         qm2rt = 'b';
         ferase = 'N';
         break;
    case 'h':
         qm2rt = helpis();
         break;
    default:
         msgerl();
         break;
    }  /* end case */
    nbftmp = nbufpo;
   }  /* end while */
    return(qm2rt);
}  /* end qmain2 */
```

```
prqual()  {  /* print-qualifications   */
    struct   st1tag   *poi1;

##   char *dmw;
##   int  k,  k2;

    poi1 = &st1[0];
    dmw = poi1->st1m;
    if(dmw != 'D' && dmw != 'M' && dmw != 'W')
     {
     printf("\nWarning: You have NO qualifications\n");
     goto endprq;
     }
    else
  printf("\nUp to now, ");
  printf("your qualifications are the following;\n");
    for(k = 1; k < 26; k++)  {
      k2 = k;
      dmw = poi1->st1m;
      switch(dmw)  {
      case 'D':
 printf("%d -%s              ", k, poi1->st1elm);
   if(poi1->st1c > 0)
     printf("     %d  listings\n", poi1->st1c);
           else printf(" \n");
            break;
      case 'M':
     printf("%d -%s(miscellaneous; has NO more\
      qualifications)", k, poi1->st1elm);
    if(poi1->st1c > 0)
      printf("  %d  records\n", poi1->st1c);
              else printf(" \n");
            break;
      case 'W':
    printf("%d -%s(has NO more qualifications)"\
          , k, poi1->st1elm);
      if(poi1->st1c > 0)
        printf("     %d  listings\n", poi1->st1c);
              else printf(" \n");
            break;
       default:
            k = 26;
            break;
      }  /* end case */
      poi1++;
    }  /* end for */
    if(k2 > 12) { // if too many qual. to fit screen
    printf("\n   to CONTINUE, PUSH RETURN key\ngo >\n");
      greadc();
    }
     endprq:  ;
    return;
}  /* end prqua */
```

```
disp12()  { /* display-2     */
    struct   st1tag  *ptr1;

##   char reltnm[13], *rtdpl, *fldmw;
##   int ctbl, i, fltab, ct3d, ddum;

    rtdpl = 'c'; fltab = 1;       ptr1 = &st1[0];

    fldmw = ptr1->st1m;

 //  copy stack1 element to temp relation name
  for(i=0; i < 12 ; i++) reltnm[i]=' '; reltnm[12]='\0';
    for(i = 0; i < 12; i++)    reltnm[i] = ptr1->st1elm[i];

  agai9:

while((fldmw=='D'||fldmw=='M'||fldmw=='W')&&fltab <= 2)
 {
     printf("processing qualification: ");
     for(i = 0; i < 13; i++)  printf("%c", reltnm[i]);
     printf(" \n");
##   range of w is reltnm

    switch(fldmw) {
    case 'D':
          disd(fltab,ptr1);    /*  in file encsrc.q */
          fltab++;
          break;
    case 'M': case 'W':
           if(ptr1->st1c == 0) {
             ctbl = 0;

##   range of d1 is tabled1
##   append tabled1(w.rid) where w.rid > 0
##   retrieve (ctbl = count(d1.rid) )
##   delete d1

              ptr1->st1c = ctbl;
            }
          if(fltab == 1) {
##           append to table1(w.rid) where w.rid > 0

          }
          if(fltab == 2) {
##          append to table2(w.rid) where w.rid > 0

          }
          fltab++;
          break;
    } /* end case */
    ptr1++;

 // copy a new record from stack1
    reltnm[6] = ' ';  /* bad spot */
```

```
    for(i = 0; i < 12; i++) reltnm[i] = ptr1->st1elm[i];
    flagw = ptr1->st1m;

} /* end while */

    if(fltab == 1) /* means stack1 emty */
    {
   printf("WARNING:You have no qualification to\
        display\n");
       goto enddis;
    }

    if(fltab == 2) { // table1 has records, copy to table3

##      range of t1 is table1
##      append to table3(t1.rid)
##      delete t1
    }


    if(fltab == 3) { /* table1 and table2 have records */

##      range of t1 is table1
##      range of t2 is table2
##      append to table3(t1.rid) where t1.rid = t2.rid
##      delete  t1
##      delete  t2
##      range of t3 is table3

    ct3d = cnttb3();   /*  in file  encsrc.a */
 printf("At this point reduced to *%d* listings\n", ct3d);
   if(ct3d == 0)
    {
    printf("There is no listing, which meets\
         qualifications\n");
##       delete t3
         goto enddis;
     }
         else {
##             append to table1(t3.rid)
##             delete  t3
               fltab = 2; goto again9; }
     } /* end if(fltab == 3) */

 distb3();   /*  in file encsrc.a */


  enddis:
     ;
    return(rtapl);
}  /* end displ2 */
```

```
////////////////////////////////////////////////////////////////
//  enclosure to src.c
//  includes  cnttb3(), distb3(), disa()
//  the reason this part is separate is that the
//            equel precompiler does not work well
//                                  for large programs.
#include "encsrc.c"
////////////////////////////////////////////////////////////////


delete(bfpt1)  /* delete qualification */
    int  *bfpt1;  {
    struct   st1tag   *s1p;
    struct   st2tag   *s2p;

    int   *bfpt2, i, *s1pend;
##   char   *rtrlx;

    s1p = &st1[0];    s2p = &st2[0];
    bfpt2 = bfpt1;   s1pend = s1p + 24;
    printf(" \n   -Type\n     one or more of the\
  qualification numbers which have been\nprinted\
  on the screen just before, separating by commas\n\
   OR\n  \"n\"  NOT want to DELETE\n");
    rtrlx = greadn(bfpt2);
    bfpt2 = bfpt1;
    if(rtrlx == 'e') goto endrlx; else rtrlx = 'c';

 // begin to erase corresponding stack1 elements

 //  99 means n; not relax
    while(*bfpt1 != 0 && *bfpt1 != 99) {
       s1p = s1p + *bfpt1 - 1; /* to adjust p. */
 // check pointer limit is not exceeded
      if(s1p > s1pend) {
      printf("\n warning: large relax argument\
         is discarded\n");
      s1p = s1p + 1 - *bfpt1; bfpt1++;  continue;}
      for(i = 0; i < 12; i++)
         s1p->st1elm[i] = ' ';
      s1p->st1m = 'X';
      s1p = s1p + 1 - *bfpt1; /* to adjust pointer */
      bfpt1++;
    }  /* end while */

    if(bfpt1 == bfpt2 && bfpt2 == 99)  goto endrlx;
 // to compress stack1 elemets
 // copy stack1 elements to stack2 (except ones X marked)
    while(s1p <= s1pend) {
     if(s1p->st1m=='D' || s1p->st1m=='M' || s1p->st1m=='W')
       { for(i = 0; i < 12; i++)
            s2p->st2elm[i] = s1p->st1elm[i];
         s2p->st2m = s1p->st1m;
```

```
            s2p->st2c = s1p->st1c;
            s2p++;
          } /* end if */
          s1p++;
      } /* end while */
  // second; copy stack2 to stack1
      cps2s1();
      endrlx:
            ;
return(rtrlx);
}    /* end delete(bfot1)  */



 ////////////////////////////////////////////////////////////////
 //  com.c is common to search, and Add-Delete-Modify;
 //  includes cylbrc, helpis, greadc, greadn, msgerl
#include "com.c"
 ////////////////////////////////////////////////////////////////
```

```
 //  This  file is an enclosure to src.a
 //  src.a contains search part of ISIS


cnttb3() {  /* count-table3:  */
 //  called from file src.a function displ2()
##  int abx;
##  range of t3 is table3
##  retrieve (abx = count(t3.rid) ) {
   ;
##   }
return(abx);
} /* end cnttb3() */



distb3() {  /*  display-table3: */
 //  called from file src.a function displ2()
##   char hrdes[51], hrname[13], *sorc, *rewhe;
##   int hrid, ct3;
    ct3 = 0;*sorc = '4';
##   range of t3 is table3
##   retrieve (ct3 = count(t3.rid) )
    if(ct3 == 0) {
   printf("There is no listing which\
    meets qualifications\n"); goto endt3;  }
     else {
       printf("There are *%d* listings which meet\
          qualifications\n", ct3);
 printf("\n");
 printf("-Type one of the following numbers in \" \"\n");
 printf(" \"1\" to SEE listings which meet\
   qualifications\n\   "2\" to RETURN and CHANGE your\
   qualifications\nqo >\n");
      while(sorc != '1' && sorc != '2') {
          sorc = greadc();
          if(sorc != '1' && sorc != '2')
          printf("WRONG argument: type 1 or 2 only\n");
       }
   }
      if(sorc == '2') goto endt3;
    printf("ID #   record name          description\n");
##   range of r is recordfile
##   retrieve(hrid=r.rid,hrdes=r.rdes,hrname=r.rname)
##   where r.rid = t3.rid {

 printf(" %d %s %s %s %s\n",hrid,"|",hrname,"|",hrdes);
##   }
   prwhr:  /* print where info. */
```

```
    printf("\n");
    printf("-Type one of the following numbers in \" \"\n");
    printf("   \"1\"  to see WHERE information to find\
    a specific record\n      \"2\"  to RETURN\ngo >\n");
     rewhe = greadc();
     if(rewhe == '2')  goto  endt3;
     if(rewhe != '1' && rewhe != '2') {
       printf("wrong argument; discarded and RETURNED\n");
       goto endt3;
      }
 //   if rewhe == '1' to see record where
     printf("   -Give only one of the ID numbers which have\
      been printed\ngo >\n");
     hrid = getnm();
##    range of wr is whr
##    retrieve(hrdes = wr.rdes) where wr.rid = hrid {
     printf(" ID #          where\n");
     printf("  %d          %s\n", hrid, hrdes);
##    }
     goto prwhr;
  endt3:
       ;
##    delete t3
      ;
  } /* end distb3() */



getnm()  {  /* get-number:  */
##    int  pn, p1, p2;
  gtbg:
     p2 = 0;

     while((pn = getchar()) != '\n')  {
       if(pn != ' ') {
          pn = pn - '0';
          if(pn >= 0 && pn <= 9) {
            p1 = pn;
            p2 = p2 * 10 + p1;
            }
       else {
       printf("wrong argument; try again\ngo >\n");
       while((pn = getchar()) != '\n') // erase input buffer
          goto gtbg;
           }
       }
    } /* end while */
return(p2);
} /* end getnm(); */
```

```
disd(fltb,prs1)    /* display-d transaction */
 //  called from file src.c function displ2()
   int  fltb, *prs1;  {

##  char rtnm[13];
##  int  k6, cloop, ctd2;

 //  copy stack1 element to temp relation name
 for(k6=0; k6 < 12; k6++) rtnm[k6]=' '; rtnm[12]='\0';
 for(k6 = 0; k6 < 12; k6++) rtnm[k6] = prs1->stlelm[k6];
##  range of r1 is rtnm
##  append to tabled1(r1.rid)   where r1.rid > 0
##  append to tabled2(r1.crname) where r1.rid = 0
##  range of d2 is tabled2
##  range of d1 is tabled1

 agai8:
##  retrieve (ctd2 = count(d2.crname) )
   if(ctd2 > 0) {
     cloop = 0;
     while(cloop < ctd2) {
##       retrieve (rtnm = d2.crname) {
         ;
##        }
##       range of s1 is rtnm
##       append to tabled1(s1.rid)  where s1.rid > 0
##       append to tabled2(s1.crname) where s1.rid = 0
##       delete d2 where d2.crname = rtnm
         cloop++;
    } /* end while */
   } /* end if */


##  delete d2
    if(fltb == 1) {
##     append to table1(d1.rid)
      }
      else {
##     append to table2(d1.rid)
      }
    if(prs1->stlc == 0) {
##    retrieve(ctd2 = count(d1.rid) )
      prs1->stlc = ctd2;
    }

##    delete d1
}  /* end disd(); */
```

```
////////////////////////////////////////////////////////////////
//  com.a contains common functions for
//                        1. SEARCH program
//                        2. ADD-DELETE-MODIFY program
////////////////////////////////////////////////////////////////



cylbrc() { // cycle-branch evaluates result of first usage
           // and sets return value to exit or begin ISIS
##    char *cc;
##    int  nc;
     nc = 0;
  again1:
   printf(" \n");
   printf("-Type one of the following numbers in \" \"\n");
   printf("  \"1\"  to EXIT ISIS and return to UNIX\n\
     \"2\"  to BEGIN ISIS again\ngo >\n");
    cc = greadc();
    switch(cc) {
    case '1':
          cc = 'e';
          break;
    case '2':
          cc = 'b';
          break;
    case 'h':
          printf("   -Now, you have only two choices \n");
          goto again1;
    default:
          if ( nc < 3 )    { nc++; msger1(); goto again1; }
             else  {  cc = 'e'; break;  }
   }
    return(cc);
}           /*  end of cylbrc routine  */
```

```
helpis() { // help-ISIS: prints information
##    char *ch;
##    int  nh;
     ch = 'c';  nh = 0;
    again5:
    printf(" \n  ");
    printf("Type one of the following numbers in \" \"\n");
    printf("  \"1\"  to EXIT ISIS and return to UNIX\n\
      \"2\"  to BEGIN ISIS at the beginning again\n\
      \"3\"  to CONTINUE as you are\n");
     if (ch != 'r')
     printf(" \"4\" to see more INFORMATION about ISIS\n");
     printf("go >\n");
     ch = greadc();
     switch(ch)  {
     case '1':
           ch = 'e';
           break;
     case '2':
           ch = 'b';
           break;
     case '3':
           ch = 'c';
           break;
     case '4':
           infohe();
           ch = 'r';  goto again5;
     default:
           if (nh < 3)  { nh++; goto again5; }
              else { ch = 'e'; break; }
    }
     return(ch);
}         /* end of helpis() */



infohe() {
printf("   This program is an online software ");
printf("dictionary\n  There are two major programs:\n");
printf("1. SEARCH program;\n   User can find existing\
software by conversing with the computer.\n");
printf("   2. ADD, DELETE, MODIFY program;\n\
   User can advertise his own software by listing\
it with the system.\n");
printf("   Provisions for modifying and deleting\
a listing are included.\n"); printf("   \nBoth programs are
easy to use.\
   You do not need previous experience.\
  Just\ntype answers after \"go >\" marks.\n");
printf("   The programs are writtenin the data\
 sublanguage QUEL, supported by the INGRES \ndata base\
 management system and the programming language C.\n\n");
     return;
}
```

```
 greadc() { // reads max 10 chars, returns first only
##    char cbuf[10], *cp, *ctemp;
     cp = &cbuf[0];
     while( (ctemp = getchar() ) != '\n')    {
       if (ctemp != ' ') { *cp = ctemp; cp++; }
     }
     return(cbuf[0]);
}



 greadn(nar) // reads max 12 numbers and/or letters
       // separated by commas; puts into array.
       // max number value is 97
     int *nar; {
##   char  *crd;  /* return from gread() */
##   int ntemp1, ntemp2, fltime, nel, cnt, nzero, nagain;
     int  *nartmp;
     nagain = 0;
     nartmp = nar;
  again:
     nar = nartmp; // to adjust pointer at endloop
     fltime = 1;
     nel = 0;
     crd = 'g';  /* if no error return 'g' for go */
          /* put zeros is array */
     for (nzero = 0; nzero < 12; nzero++)
       { *nartmp = 0; nartmp++; }
      nartmp = nartmp - 12;  /* adjust nartmp to point */
            /* to beginning of the array to use again  */

     printf("go >\n");
     while( (ntemp1 = getchar()) != '\n' ) {
         if(ntemp1 != ' ') {
        // if input is "n":
        // put it in array, erase input buffer, return
                if(ntemp1 == 'n') {
                  *nar = 99;
      while((ntemp1 = getchar()) != '\n') ; /* null stm. */
                  ntemp1 = ' ';
                  goto endloop;
                } /* end if */

  // if input "m";put in array, continue other arguments
            if(ntemp1 == 'm') {
              *nar = 98;
              nar++;
           while((ntemp1 = getchar()) == ' ') ;/*null stm.*/
              if(ntemp1 == ','){ntemp1 = ' '; continue;}
              if(ntemp1 == '\n') goto endloop;
               else {
                 printf("wrong argument after \"m\"\n");
                 crd = 'e';
```

```
                goto endloop;
                    }
                }

        /* input should be number, so go ahead */
          ntemp1 = ntemp1 - '0';

        switch(fltime) {
        case 1:
             cnt = 0;
             if(ntemp1 >= 0 && ntemp1 <= 9)
               {ntemp2 = ntemp1; fltime = 2; continue;}
              else {
             printf("wrong argument; expecting number\n");
             crd = 'e'; goto endloop;
                   }
        case 2:
             cnt++;
             if(cnt > 1 && ntemp1 != (',' - '0') ) {
                 crd = 'e';
             printf("wrong argument; very large\n");
              goto endloop;
              }
            if(ntemp1 >=0 && ntemp1 <= 9)
              { ntemp2 = ntemp2 * 10 + ntemp1; continue; }
             else  if(ntemp1 ==  (',' - '0') )
                     { fltime = 1; break; }
              else {
              printf("wrong argument;");
              printf(" expecting number or \",\"\n");
              crd = 'e'; goto endloop;
                   }
       }  /*  end switch  */
       if(ntemp2 >= 1 && ntemp2 <= 97) {
         *nar = ntemp2; ntemp2 = 0; nar++;
       }
        else {
        printf("wrong argument; very large or less than 1\n");
        crd = 'e'; goto endloop;
             }
        nel++;
        if(nel > 11) {
        printf("more than 11 arguments\n"); crd = 'e';
        goto endloop;
          }
     }  /*  end if(ntemp1 != ' ') */
   }  /* end while */
    if(ntemp2 >= 1 && ntemp2 <= 97) {
     *nar = ntemp2; ntemp2 = 0;
    }
    else {
     printf("wrong argument; very large or less than 1\n\
       or ending with \",\" without argument\n");
     crd = 'e';
```

```
      }
endloop:
     nagain++;
      if(nagain < 3 && crd == 'e')  {
       printf(" \n  -Type one or more of the following\
       numbers in \" \"\n      separated by commas\n
       Try one more time\n");
         if(ntemp1 != '\n')       /* to erase buffer */
          while((ntemp1 = getchar()) != '\n')     ;
           goto againr;
      }
return(crd);
}   /*  end greadn(nar)  */



msger1() {
     printf("  You must type one of the characters in \
 \" \" which have been printed on the\n terminal\
 or type \"h\" to get help information\n");
}
```

```
//  Thesis about INGRES data base application ISIS
//  This program is ADD-DELETE-MODIFY part of ISIS
//  This file has two enclosures:  1. encadm.c   2. com.c




extern  char genrel[13];
extern  int  idnum;
extern  char flname[13],fldes[51],fldm,flwhr[51],passw;



main()  { /* add-delete-modify */
    char  is0or1;
    infohe();
    printf("\n\n  This part of program does ADD, MODIFY\
     and DELETE functions\n");
  aga01:
    printf("  -Type one of the following in \" \"\n\
    \"1\"  to begin ACTUAL usage\n\
    \"2\"  to begin DEMONSTRATION usage\nGo >\n");
    is0or1 = greadc();
    if(is0or1 == '1') {
##   ingres isis0  // invokes ingres database named ISIS0

    }
    if(is0or1 == '2') {
##   ingres isis1  // invokes demonstration database ISIS1
    }
    if(is0or1 != '1' && is0or1 != '2')
    { printf("\nWrong argument, try again\n");  goto aga01;
}
    branch();
    printf("ISIS add-delete-modify program has been logged\
     out\nGoodby, come again\n");
##   exit
}
```

```
branch() { /* branches to one of the main routines */
    extern  char fldm;
##   char  *c, *rt2;
##   int   n;

    n = 0;   rt2 = 'e';
  cylbr:
    printf("\n  -Type one");
    printf(" of the following numbers in \" \"\n");
    printf("      \"1\"  to ADD a new listing for a new\
    program\n\"2\"  to MODIFY an existing listing\n\
    \"3\"  to DELETE all information about one program\n");
  again:
    printf("go >\n");
    c = greadc();
    switch(c)  {
    case '1':
          rt2 = add();
          break;
    case '2':
          fldm = 'M';
          rt2 = delmod();
          break;
    case '3':
          fldm = 'D';
          rt2 = delmod();
          break;
    case 'h':
          rt2 = helpis();
          break;
    default:
          if(n < 3) { msgerl(); n++; goto again; }
            else  break;
   } /* end switch */
    if(rt2 == 'f')  rt2 = cylbrc(); /* f means finish */
    if(rt2 == 'b' || rt2 == 'c') goto cylbr;
    return;
} /* end branch(); */



add() {
## char *rta;
    rta = 'f';
    printf("   ADD in progress ...\n");
    getpar()      /*  get-parameters */
    aprfwh();     /*  append-recordfile and where rel. */
    rta = addrls();  /* add-to-relations */
    if(rta == 'e')  goto endadd;
  endadd:
      ;
    return(rta);
}  /* end add() */
```

```
getpar() { /* get-parameters for adding */
    extern int idnum;
    extern char flname[13], fldes[51], flwhr[51], passw;
##   int  tmax, i;
##   char *chtmp, *flwt;
 //  initialization of variables
     idnum = 0; tmax = 0;
     flwt = 'a'; flname[12] = '\0'; fldes[50] = '\0';

##   range of r is recordfile
##   retrieve  (tmax = max(r.rid))
    idnum = tmax + 1;
    printf("  Very important !  Keep in mind\n\
    Your new record ID number is  * %d *\n", idnum);

 //  erase flname and fldes to begin clean
    for(i = 0; i < 12; i++)  flname[i] = ' ';
    for(i = 0; i < 50; i++)  fldes[i] = ' ';

 //  accept input from terminal
   printf("  -Give new PROGRAM'S NAME");
   printf(" (NOT more than 12 characters)\n");
   printf("do not insert blanks between characters\n");
   printf("go >\n");
    i = 0;
    while( (chtmp = getchar() ) != '\n')  {
      if(flwt == 'a') {
        if(chtmp != ' ') { flname[i] = chtmp; i++;}
      }
      if(flwt == 'a' && i > 12)  {
        printf("long argument; truncated\n");
        flwt = 's';
      }
     } /* end while */
    flname[12] = '\0';
    flwt = 'a';

   printf("  -Give new PROGRAM'S DESCRIPTION (NOT\
    more than 50 characters, in one line)\ngo >\n");
    i = 0;
    while( (chtmp = getchar() ) != '\n')   {
      if(flwt == 'a')  fldes[i] = chtmp;
      i++;
      if(flwt == 'a' && i > 50) {
        printf("long argument; truncated\n");
        flwt = 's';
      }
    } /* end while */
    fldes[50] = '\0';
 //  get where rdes input from terminal
  printf("  -Tell WHERE this program can be found\n\
   (NOT more than 50 characters\ngo >\n");
    flwt = 'a';
    i = 0;
```

```
    while((chtmp = getchar()) != '\n') {
       if(flwt == 'a')  flwhr[i] = chtmp;
       i++;
       if(flwt == 'a' && i > 50) {
         printf("long argument; truncated\n"); flwt = 's';
          }
     } /* end while */
    printf("  -Give one character \n");
    printf("to be new listing's PASSWORD\n");
    printf("go >\n");
    passw = greadc();

} /* end getpar() */



aprfwh() { /* append-recordfile append-whr rel. */
  extern  int  idnum;
  extern  char flname[13], fldes[51], flwhr[51], passw;
##   int cmax, k;
##   char fname[13], fdes[51], fwhr[51], pss[2];

    cmax = idnum;  pss[0] = passw;  pss[1] = '\0';

    for(k = 0; k < 12; k++)  fname[k] = flname[k];
    for(k = 0; k < 50; k++) {
      fdes[k] = fldes[k];  fwhr[k] = flwhr[k];
     }
## append recordfile(rid=cmax,rname=fname,rdes=fdes,
##                   pswrd = pss)
## append whr(rid = cmax, rdes = fwhr)

 } /* end aprfwh */
```



```
////////////////////////////////////////////////////////////////

    struct   strtag {
##                  char  strelm[13];
                    } str[25];

////////////////////////////////////////////////////////////////
```

```
addrls()  { /* add-to-relations:  */
    struct  strtag  *pstr;
    extern  char  genrel[15];

##    int  mnum;
##    char mname[13], *rtadrl, *flfin, *rtgr, *rtcur;
      int  buf[12], *pb, j, jl, *tp;
    pstr = &str[0]; mname[12] = '\0'; flfin = 'n';
    rtadrl = 'f';
 //  to use stackr again without exiting program,
 //  initialize empty
    for(j = 0; j < 25; j++) {
       for(jl = 0; jl < 15; jl++)  pstr->strelm[jl] = ' ';
       pstr++;
    } /* end 1.for */

 //  begin display maintable and get input
   printf(" -To ADD your new listing in the appropriate\
    categories,\n");
   printf("  Type one or more of the following\
    numbers in \" \"\n    separating by commas\n\
    You MUST type at least one\n");
   printf("    Select all that pertains;\
     you will get no second chance\n");
 //  a) display maintable
##    range of m is maintable
    mnum = 0;
##    retrieve (mname = m.crname) {
    mnum++;
      printf("  \"%d\"   %s\n", mnum, mname);
##    }

 //  b) get input from terminal and put to array
    rtgr = greadn(&buf[0]);
    if(rtgr == 'e')  { rtadrl = 'e'; goto endadr; }
    genrel[0] = 'm'; genrel[1] = 'a'; genrel[2] = 'i';
    genrel[3] = 'n'; genrel[4] = 't'; genrel[5] = 'a';
    genrel[6] = 'b'; genrel[7] = 'l'; genrel[8] = 'e';
    genrel[9] = ' '; genrel[10] = ' '; genrel[11] = ' ';

    putstr(&str[0],&buf[0]);  /* put relations to stackr */

    ////////////////////////////////////////////////////////////
    while(flfin == 'n')  {  /* means not finish */
 //  dumwrt() /* dummy-write */
 //  find last element on stack to get, if emty finish
    pstr = &str[0];
    if(pstr->strelm[0] == ' ')
      { flfin = 'y';  goto endwhi; } /* stack is emty */
    j = 0;
      while(pstr->strelm[0] != ' ') {
          j++;  pstr++;
          if(j > 25) {
             printf("ERROR:adding stack overflow");
```

```
                printf(", max 25 items.\n");
                goto endwhi;
               }
            } /* end while */
      pstr = pstr - 1;

 //  copy the last stack element to genrel[13]
      for(j = 0; j < 12; j++)  genrel[j] = pstr->strelm[j];
 //  call pcurcl()  to do branch
      rtcur = pcurcl();
 //  branch  according to return of pcurcl()
      switch(rtcur)  {
      case '1':
            addcur();  /* add to current class name */
            for(j = 0; j < 12; j++) pstr->strelm[j] = ' ';
            break;
      case '2':
            newbrc();  /* new-branch: */
 //  erase one element from stack
            for(j = 0; j < 12; j++) pstr->strelm[j] = ' ';
            break;
      case '3':
            exibrc(&buf[0],pstr);  /* existing-branch: */
            break;
      case 'n':
 //         erase  one element from stack
            for(j = 0; j < 12; j++) pstr->strelm[j] = ' ';
            goto endwhi;
      } /* end switch */
  endwhi:
      if(flfin == 'y')
         printf("\n ADD process has been completed\n");
      } /* end while */
      ////////////////////////////////////////////////////////////

  endadr:   ;
return(rtadrl);
/* end addrls() */


dumwrt() { /* DUMMY stackr write */
  struct strtag *p5;
  int y, yl;
 //  DUMMY write
      p5 = &str[0];
      for(y = 0; y < 5; y++) {
         printf("%d ",y);
         for(yl = 0; yl < 12; yl++)
            printf("%c", p5->strelm[yl]);
         p5++;
      } /* end dummy write end for l. */
      printf(" \n");
} /* end dumwrt() */
```

```
putstr(pst,pbf)   /* put relation on stackr */
 //  must adjust pointers before call putstr
  int  *pst, *pbf;    {
    extern  char  genrel[13];
    struct  strtag  *psta;

##   char  arel[13], *flwrt, aname[13], *flrun;
     int   i, mnum;
    psta = pst;
        aname[12] = '\0';
    for(i = 0; i < 12; i++)
      arel[i] = genrel[i];  arel[12] = '\0';
##   range of a is arel
    flrun = '1';
    while(*pbf != 0)  {
      mnum = 0;

##      retrieve (aname = a.crname) {
      mnum++;
      flwrt = 'n';
      if(*pbf == mnum) {
        for(i = 0; i < 12; i++) psta->strelm[i] = aname[i];
        psta->strelm[12] = '\0';
        psta++; pbf++; flwrt = 'y';
        } /* end if */
##     }

      if(flwrt == 'n')
        if(flrun == '1')  flrun = '2';
          else if(*pbf > 0) {
           printf("wrong argument %d discarded\n", *pbf);
           flrun = '1'; pbf++;
          }
    } /* end while */

return;
}  /* end putstr() */
```

```
pcurcl()  {  /* print-current-class-and-branches */
    extern  char  genrel[13];
##   char  grel1[13], rel2[13], *n123;
##   int   a1, flw1, flex;
    n123 = 'n'; flex = 'n';
    for(a1 = 0; a1< 12; a1++) grel1[a1] = genrel[a1];
    grel1[12] = '\0';
    printf("\nCURRENT CLASS NAME is  * %s *\n", grel1);
    for(a1 = 0; a1 < 12; a1++)  rel2[a1] = ' ';
    flw1 = 0;
##   range of g is grel1
##   retrieve (rel2 = g.crname) where g.rid = 0 {
       flw1++;
       if(flw1 == 1) {
         printf("       and SUBCATEGORIES are:\n");
         flex = 'y';
         }
       if(flw1 > 0)  printf("           %s\n", rel2);
##   }
    printf("  -Type only one of the");
    printf("following characters in \" \"\n\ to ADD\
      your listing\n");
    printf("     \"1\"  into CURRENT CLASS NAME\n\
    \"2\"  make NEW SUBCATEGORY and put this into it\n");
    if(flex == 'y')
    printf(" \"3\" use an EXISTING SUBCATEGORY\n");
    printf("     \"n\"  None of the above;STOP\
      this direction\ngo >\n");
    n123 = greadc();
return(n123);
}  /* end pcurcl() */


addcur()  { // add-current: add to current record name
    extern   char  genrel[13],  flname[13];
    extern   int   idnum;
##   char  grel3[13],  name3[13];
##   int   idnum3,    i3;
    grel3[12] = '\0';  name3[12] = '\0';
 //  copy extern to local names
    for(i3 = 0; i3 < 12; i3++)
       { grel3[i3] = genrel[i3];  name3[i3] = flname[i3]; }
    idnum3 = idnum;
 //  add record to current name
##    append to grel3(rid = idnum3, crname = name3)
return;
}  /* end addcur() */
```

```
newbrc() { // new-branch: adds record to new branch
    extern   char  genrel[13], flname[13];
    extern   int   ianum;

##   char gra4[13],a4name[13],f4name[13],*a4char,*a4wrt;
##   int   a4, ia4num;

    gra4[12] = '\0';  a4wrt = 'g';
 //  print current class name again
       //  copy genrel to local gra4
       for(a4 = 0; a4 < 12; a4++)  gra4[a4] = genrel[a4];
    printf("  CURRENT CLASS NAME is  * %s *\n", gra4);
    printf("    NEW SUBCATEGORY will be a branch of\
      the CURRENT CLASS NAME\n");
    printf(" -Give NEW SUBCATEGORY NAME\
     (not more than 12 characters)\n");
    printf("  do NOT insert BLANKS between char.s\n");
    printf("go >\n");
  agatr:

 //  get new name from terminal
    for(a4 = 0; a4 < 12; a4++) a4name[a4] = ' ';
    a4 = 0;
    while( (a4char = getchar() ) != '\n') {
      if(a4wrt == 'g') {
         if(a4char != ' ') {
             if((a4char < 'a' || a4char > 'z') &&
                (a4char < 'A' || a4char > 'Z') &&
                (a4char < '0' || a4char > '9') &&
                 a4char != '←')
          {
      printf("wrong argument;\"%c\",try again\n",a4char);
        while((a4char = getchar() ) != '\n') ; /*erase*/
        printf("go >\n"); goto agatr;
        }
           a4name[a4] = a4char; a4++;
      }
    }
       if(a4wrt == 'g' && a4 > 12) {
        printf("long argument; truncated\n");
        a4wrt = 's';
       }
    } /* end while */
    a4name[12] = '\0';  a4wrt = 'g';
 //  create new relation with new branch name; a4name
##    create a4name(ria = i2, crname = c12)
      ;
 ////////////////////////////////////////////////////////////
 // New relations must be SAVEd to prevent automatic
 // removed by INGRES. The SAVE command date must be
 // advanced periodically  to  avoid  inadvertant  loss  of
data.
 ##   save a4name until jun 30 1978
 ////////////////////////////////////////////////////////////
```

```
 // append record to new relation a4name
    //  copy parameters to local
     id4num = idnum;
     for(a4 = 0; a4 < 12; a4++)  f4name[a4] = flname[a4];
     f4name[12] = '\0';
##   append to a4name(rid = id4num, crname = f4name)
    ;
 //   also append to current class name
##   append to gra4(ric = 0, crname = a4name)
    ;
return;
} /* end newbrc() */




exibrc(pba1,psa1)
     /* existing-branch: adds record to existing branch */
    int *pba1, *psa1; {
    extern   char  genrel[13];
    struct   strtag    *psa2;

##   char  grla[13], ganame[13];
     int   a3, *pba2;

    grla[12] = '\0';  psa2 = psa1;  pba2 = pba1;

 //  print  branches again
       // copy genrel to local
       for(a3 = 0; a3 < 12; a3++) grla[a3] = genrel[a3];

    printf(" -Type one or more of the following\
      numbers in \" \" to select subcategory\n");

##   range of ga is grla
    a3 = 0;
##   retrieve(ganame = ga.crname) where ga.rid = 0 {
    a3++;
    printf("    \"%d\"  %s\n", a3, ganame);
##   }


    greadn(pba1);

 //  remove one top element from stack
    for(a3 = 0; a3 < 12; a3++)  psa2->strelm[a3] = ' ';

 //  put selected branch names on to stack
     //  adjust genrel
        for(a3 = 0; a3 < 12; a3++)  genrel[a3] = grla[a3];
```

```
        //  put relations to stack, pass stack and buf. poin.
        putstr(osa2,pba2);
    return;
    } /* end exibrc() */


    ////////////////////////////////////////////////////////////////////
    //  includes delmod(), getnum(),  delete(), modify()
    #include "encadm.c"


    // com.c includes; cylbrc(), helpis(), greadc(), greadn(),
msgerl()
    #include "com.c"
    ////////////////////////////////////////////////////////////////////
```

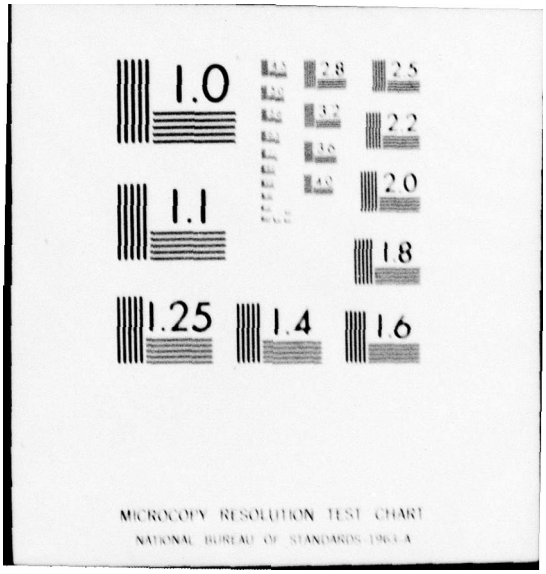
```
//  This file is an enclosure to file adm.q
//  File adm.q is the Add-Delete-Modify part of ISIS



delmod()  {
   /* delete-modify: does common actions, and branches */
    extern  char  fldm;
    extern  int   idnum;

##   int tlid, t2id, timen, flow;
## char *rtdm,*flfind,*flgo,rcname[13],rcdes[51];
## char psw1[2], psw2[2];
    timen = 0;
  beadm:
    rtdm = 'f'; rcname[12] = '\0'; rcdes[50] = '\0';
    rtdm = 'f';
  agadm:
    printf("  -Give ID NUMBER of your record, which you\
    are going to"); if(fldm == 'D') printf(" delete\n");
              else printf(" modify\n");
     printf("go >\n");
    t2id = 0;
##   range of r is recordfile
    tlid = getnum();  /* get input and compute ID # */
    t2id = 0; flfind = 'n';
##   retrieve (t2id = r.rid) where r.rid = tlid {
     ;
##   }
    if(t2id != 0) { /* means rid is correct */
    printf("  -Give only one character as a PASSWORD\n");
    printf("go >\n");
    psw1[0] = greadc();
    flow = 0;
##   retrieve (psw2 = r.pswrd) where r.rid = tlid {
     if(psw1[0] != psw2[0])
      { printf("wrong password; exited\n"); flow = 1; }
##   }
    if(flow == 1) goto enddm;
##  retrieve(rcname = r.rname, rcdes = r.rdes)
##      where r.rid = t2id  {
    printf("\nID #    record name      description\n%d\
    %s    %s\n", t2id, rcname, rcdes);
    printf("\nIs this the listing you are seeking ?\n\
    -Type \"yes\"  or  \"no\"\ngo >\n");
    flfind = 'y';
##   }
    ;
```

AD-A056 323 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 5/2
THE DESIGN AND IMPLEMENTATION OF INSTRUCTIONAL SOFTWARE INFORMA--ETC(U)
JUN 78 A YILDIRIM

UNCLASSIFIED NL

2 OF 2
AD
A056 323

END
DATE
FILMED
8-78
DDC


1.0
1.1
1.25
1.4
1.6
2.8
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

```
    } /* end if(t2id != 0) */
    timen++;
    if(flfind == 'n' && timen > 2) {
      printf("Wrong ID number three times;\
          program exited\n");
      rtdm = 'e'; goto enddm;
      }
    if(flfind == 'n') {
      printf("Try one more time\ngo >\n");
      goto begdm;
      }

 //  if finds comes here
   findgo:
    flgo = greadc();
    if(flgo == 'n')  goto enddm;
    if(flgo != 'y') {printf("Try one more time\ngo >\n");
                     goto findgo; }
    idnum = t2id;
    if(fldm == 'D')  delete();
    if(fldm == 'M')  modify();
  enddm:
    ;
return(rtdm);
}  /* and delmod() */



getnum()  {
// read char.s from terminal, computes one int. number
##   int tn, t1, t2;
  getbeg:
    t2 = 0;

    while((tn = getchar()) != '\n') {
      if(tn != ' ') {
         tn = tn - '0';
         if(tn >= 0 && tn <= 9) {
           t1 = tn;
           t2 = t2 * 10 + t1;
           }
       else{
         printf("wrong argument; try again\ngo >\n");
         while((tn = getchar()) != '\n') // erase buffer
         goto getbeg;
           }
       }
    } /* end while */
return(t2);
} /* end getnum() */
```

```
delete()  {
    extern    int   idnum;
    extern    char genrel[13];
    struct    strtag  *psd;
##   int  iddlt, f, f1;
##   char flname[13], *ffin;
    iddlt = idnum; psd = &str[0];  flname[12] = '\0';
    ffin = 'n';
 //  delete from recordfile relation
##   range of r is recordfile
##   delete r where r.rid = iddlt
 //  delete from where relation
##   range of wh is whr
##   delete wh  where wh.rid = iddlt
 //  initialize stackr emty
    for(f = 0; f < 25; f++) {
       for(f1 = 0; f1 < 25; f1++)  psd->strelm[f1] = ' ';
       psd++;
    } /* end l. for */
    psd = &str[0];
 //  load stackr with maintable relations
##   range of m is maintable
    for(f = 0; f < 12; f++)  flname[f] = ' ';
##   retrieve (flname = m.crname) {
    //  copy relation to stack and erase flname
        for(f = 0; f < 12; f++) {
          psd->strelm[f] = flname[f];
          flname[f] = ' ';
        }
        psd++;
##   }
 //  begin delete by using stack
    printf("deletion in progress ...\n");
    while(ffin == 'n') { // means stackr not finished
// dumwrt();  /* DUMMY WRITE */
  //  find last element on stack to get; if empty, finish
       psd = &str[0];
       if(psd->strelm[0] == ' ') {
         ffin = 'y';
         goto endle;
        }
       f = 0;
       while(psd->strelm[0] != ' ') {
         f++;  psd++;
         if(f > 25) {
         printf("WARNING: delete stack overflow process");
         printf(" continues, max 25 items.\n");
         psd--;  break; }
       } /* end while */
       psd--;
    //  copy last stack element to genrel[13]
       for(f = 0; f < 12; f++) {
          genrel[f] = psd->strelm[f];
       }
```

```
    //  call delete by passing stack pointer
        prodlt(psd);
    endle:
        ;
    } /* end while */
        printf("delation has been completed\n");
} /* end delete() */



prodlt(psdlt)   /* process-delete  */
 int  *psdlt;  {
    extern   char genrel[13];
    extern   int  idnum;
    struct   strtag  *psdl;

##   int  iddlt1, dl;
##   char gndlt[13], dltrel[13], *fldlt, dfname[13];
 //  intialize and copy global to local
    psdl = psdlt;  iddlt1 = idnum;
    gndlt[12] = '\0';  dfname[12] = '\0';  fldlt = 'n';
    for(dl = 0; dl < 12; dl++) {
       gndlt[dl] = genrel[dl];  dltrel[dl] = ' ';
      }
     dltrel[12] = '\0';
##   range of de is gndlt
##   retrieve(dltrel = de.crname)  where de.rid = iddlt1 {
      fldlt = 'y';
##   }
 //  if there is record with delete id # delete and return
    if(fldlt == 'y') {
##     delete  de  where de.rid = iddlt1
      dl = 1;
##     retrieve (dl = count(de.rid) )
     }
    if(dl == 0) {
##   destroy  gndlt
##   range of df is dfname
##   delete  df where df.crname = gndlt
    }
    if(fldlt == 'y')  {
      for(dl = 0; dl < 12; dl++) psdl->strelm[dl] = ' ';
      goto endpro;
     }
 //  copy stack top to dfname to use next time and delete
    for(dl = 0; dl < 12; dl++) {
      dfname[dl] = psdl->strelm[dl];
      psdl->strelm[dl] = ' ';
       }
 //  if there is a branch relation out on stack
##   retrieve(dltrel = de.crname) where de.rid = 0 {
      for(dl = 0; dl < 12; dl++)
        psdl->strelm[dl] = dltrel[dl];
```

```
        osalt++;
##    }
 endpro:
      ;
} /* end prodlt() */



modify()  {
##    char  fl12n[2], *fl1, *fl2;
      int   z, *pl2n;

      fl12n[1] = '\0';
      fl1 = '1';  fl2 = '2';
      printf("\n    It is allowed to modify only\n\
         program DESCRIPTION\n                and\n\
         program WHERE information\n");
      printf("   If you want to do other than this use\
         DELETE to remove\n the listing and\
         than use ADD to create a revised one.\n");
   begmod:
     while(fl1 == '1' || fl2 == '2') {
     printf(" \n   ");
     printf("-Type one of the following in \" \"\n");
     if(fl1 == '1')
    printf("   \"1\"  to modify record DESCRIPTION\n");
     if(fl2 == '2')
  printf("  \"2\"  to modify record WHERE information\n");
  printf(" \"n\" NOT to do anything, and RETURN\ngo >\n");
     fl12n[0] = greadc();
     pl2n = &fl12n[0];
     switch(fl12n[0])  {
     case '1':
           mdewh(pl2n);
           fl1 = '3';
           break;
     case '2':
           mdewh(pl2n);
           fl2 = '3';
           break;
     case 'n':
           goto endmod;
     } /* end switch */
   } /* end while */
   endmod:
      ;
} /* end modify() */
```

```
mdewh(pt)  /* modify-description-where-relations */
  int  *pt;  {
    extern  int idnum;
##   int  tdnum, z;
##   char  pdes[51], *tch;
    tdnum = idnum;  pdes[50] = '\0';

    if(*pt == '1') { /* means modify description */
##      range of r is recordfile
##      retrieve(pdes = r.rdes) where r.rid = tdnum {
  printf("Present program description is;\n %s\n\n",pdes);
##      }
     } /* end if */

    if(*pt == '2')  {  /* means modify where */
##      range of wh is whr
##      retrieve(pdes = wh.rdes) where wh.rid = tdnum {
  printf("Present WHERE information is;\n %s\n\n",pdes);
##      }
     } /* end if */

  if(*pt == '1' || *pt == '2') {
       printf("  -Give new listing");
       if(*pt == '1')
         printf(" DESCRIPTION (NOT more than 50 ");
       if(*pt == '2')
         printf(" WHERE information (NOT more than 50 ");
       printf("characters)\ngo >\n");
    for(z = 0; z < 50; z++)  pdes[z] = ' ';
    z = 0;
    while((tch = getchar()) != '\n') {
        pdes[z] = tch;  z++;
        if(z > 51) {
    printf("     Long line; truncated\n\
    only first 50 characters have been accepted\n");
         pdes[50] = '\0';
         break;
         }
     } /* end while */
    if(z < 51)  pdes[z] = '\0';
  } /* end if */
    if(*pt == '1') {
##      replace r(rdes = pdes) where r.rid = tdnum
    }
    if(*pt == '2') {
##      replace wh(rdes = pdes) where wh.rid = tdnum
    }

}   /* end mdewh() */
```

LIST OF REFERENCES

1. Brian W.K. , Programming in C, A Tutorial.
   Bell Laboratories,
   Murray Hill, New Jersey 07974

2. Dennis M.R. , C Reference Manual.
   Bell Laboratories
   Murray Hill, New Jersey 07974, 1974.

3. Epstein R, A Tutorial on INGRES.
   University of California, Berkeley 94720, 1977.

4. Epstein R. ,
   Creating and Maintaining a Database Using INGRES,
   Memorandum No.UCB/ERL M77/71,
   University of California, Berkeley 94720, 1977.

5. Stonebraker M., Allman E. and Held G.
   Embedding a Relational Data Sublanguage
   In a General Purpose Programming Language,
   Memorandum No.ERL/M564,
   University of California, Berkeley 94720, 1975.

6. Yousseti K. , Whyte N. , Ubell M. , Ries D. ,
   Hawthorn P. , Epstein B., Berman R., Allman E.,
   INGRES Reference Manual Version 6
   Memorandum No.ERL/M579
   Electronics Research Laboratory
   University of California, Berkeley 94720, 1977.

INITIAL DISTRIBUTION LIST

1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Asst Professor S. T. Hall, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
5.	Deniz Kuvvetleri Komutanligi Egitim Daire Baskanligi Ankara V TURKEY	5
6.	Lt. Ataman YILDIRIM Findikzade Tevfik Fikret sok. No.62 D.15 Topkapi Istanbul Z TURKEY	2
7.	Department of Computer Science Naval Postgraduate School Monterey, California 93940	6